Ilari Juvani

WordPress-kehittämisen hyvät käytänteet

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Mediatekniikan koulutusohjelma

Insinöörityö

19.3.2018

Metropolia

Tekijä Otsikko Sivumäärä Aika	Ilari Juvani WordPress-kehittämisen hyvät käytänteet 62 sivua 19.3.2018
Tutkinto	insinööri (AMK)
Koulutusohjelma	mediatekniikka
Suuntautumisvaihtoehto	digitaalinen media
Ohjaaja	lehtori Ilkka Kylmäniemi

Insinöörityössä perehdyttiin WordPress-julkaisujärjestelmään verkkokehittämisen ja verkkosovelluskehyksien näkökulmasta. Työssä arvioitiin ja pohdittiin tämän suositun verkkosisällönhallintajärjestelmän soveltuvuutta ammattimaisten ja modernien verkkosivustojen kehittämiseen ja tuotantoon. Hyvän verkkosovelluskehyksen tunnuspiirre on runsas määrä dokumentoituja työkaluja ja rajapintoja, joilla abstrahoidaan verkkokehityksessä usein toistuvia toimenpiteitä, kuten käyttäjätietojen hallinnointi, tietokantayhteyksien muodostaminen ja datan manipulointi sekä syötedatan validointi. Keskeisenä lähtökohtana ovat luonnollisesti myös erilaiset uudelleenkäytettävät komponentit, joita kehittäjät voivat hyödyntää rakentaessaan monimutkaisempaa logiikkaa verkkosivustoille. Havaittiin, että WordPressin merkittävimpiä vahvuuksia ovat alustan joustavuus ja laajennettavuus, joiden taustalla toimii lisäosarajapinta. Työssä tarkasteltiin, kuinka teema- ja lisäosakehittäjät voivat hyödyntää tätä rajapintaa rakentaessaan mukautettua toiminnallisuutta WordPress-sivustoille.

Insinöörityön keskeinen tarkoitus oli lisäksi tutkia erilaisia verkkokehittämisen menettelytapoja, jotka voidaan perustella hyviksi käytänteiksi. Näitä verkkokehittämisen käytänteitä tarkasteltiin aluksi yleisellä tasolla, minkä jälkeen käsittelyä rajattiin tarkemmin koskemaan WordPress-kehittämisen käytänteitä. Hyvien käytänteiden osalta työssä perehdyttiin kehitysympäristöihin ja versionhallintaan, suorituskyvyn optimointiin, responsiivisuuteen sekä WordPress-teemakehityksen standardeihin ja ohjelmoinnin käytänteisiin. Insinöörityössä tehtiin myös full-stack-luontoinen verkkokehitystyö helsinkiläiselle mainostoimistolle. Projektissa rakennettiin ja toimitettiin yrityksen WordPress-pohjaisille kotisivuille räätälöity teema yrityksen visuaalisen suunnittelutiimin työstämien layoutien ja toiminnallisten määrittelyjen pohjalta. Projektin aikana pyrittiin mahdollisimman laajasti soveltamaan WordPress-teemakehityksen standardeja ja hyviä käytänteitä. Noudattamalla hyviä käytänteitä projektissa varmistettiin, että verkkosivuston kehittämisen työnkulku säilyi jouhevana, ja lopputuote oli helposti muokattavissa eikä sisältänyt riskialtista koodia.

Avainsanat	WordPress, verkkokehitys, verkkosovelluskehys

Author Title Number of Pages Date	Ilari Juvani WordPress development best practices 62 pages 19 March 2018
Degree	Bachelor of Engineering
Degree Programme	Media Technology
Specialisation option	Digital Media
Instructor	Ilkka Kylmäniemi, Senior Lecturer

The thesis examines WordPress platform from the perspective of web development and web application frameworks. One of the main purposes of the thesis was to evaluate the adequacy of WordPress for developing and producing modern and professional websites. One of the most important features of a good quality web application framework is the abundance of well documented tools and application programming interfaces, which abstract various repeated processes in web development, such as user data management, establishing database connections, manipulating queried data and validating user input data. An important observation is also the availability of reusable components that developer can utilize as he or she builds more complex logic for the website. The research identified the plugin API as one of the core factors behind the high flexibility and extensibility of WordPress. The thesis examines how one can utilize the plugin API to build advanced and customized functionality for WordPress websites.

An important objective was also to study various methodologies considered as best practices in web development. These practices are first examined broadly from the view of web development and later narrowed down more precisely to WordPress development. Within the topic of best practices, the thesis considers development environment and revision control, performance optimization, responsiveness, standards of theme development and WordPress coding practices. The thesis project was conducted as a full-stack orientated web development task for Helsinki city-based advertising agency. The goal of the project was to develop and deliver a customized WordPress theme to be used in the company website, based on the layouts and functional specifications composed by the internal design team. Development work was conducted with high respect and consideration towards WordPress theme development standards and coding best practices. Conforming to the standards and best practices assured rapid development cycles as well as easy modification and risk-free code of the end product.

Keywords	WordPress, web development, web application framework

Sisällys

Lyhenteet

WWW	World Wide Web. Internetin selattava osa, verkko, jonka käyttöliittymänä toimii yleensä verkkoselain.
FTP	File Transfer Protocol. Standardiprotokolla, jolla voidaan siirtää tiedostoja asiakas- ja palvelinkoneen välillä TCP/IP-yhteyden välityksellä.
PHP	PHP Hypertext Preprocessor. Palvelinpään ohjelmointikieli, jolla tuotetaan dynaamisia verkkosivuja.
HTTP	Hypertext Transfer Protocol. OSI-mallin sovelluskerroksen tilaton protokolla, jolla välitetään hypermediadokumentteja, tai muuta dataa, verkossa asiakas- ja palvelinkoneen välillä.
ASP	Active Server Pages. Microsoftin kehittämä dynaamisten verkkosivujen luontiin tarkoitettu palvelinpään skriptikieli.
MVC	Model-View-Controller. Sovellusarkkitehtuurimalli, jossa sovelluksen toiminta on jaettu malli-, näkymä- ja ohjainkerroksiin.
URL	Uniform Resource Locator. Merkkijono, jolla määritellään yksikäsitteisesti tietyn verkkodokumentin tai -aineiston sijainti verkkopalvelimella.
API	Application Programming Interface. Sovellusrajapinta, joka kapseloi sisälleen logiikkaa, metodeita tai muita resursseja, joita kolmannen osapuolen sovellukset voivat kutsua tarvittaessa.
LAMP	Linux-Apache-MySQL-PHP/-Perl/-Python-palvelinsovelluspino. LAMP-pinoon pohjautuva verkkosovellus käyttää kaikkia edellä mainittuja verkkoteknologioita (viimeisenä mainituista skriptikielistä käytetään yleensä vain yhtä, vaikka useamman kuin yhden yhtäaikainen käyttö ei ole poissuljettua).
SQL	Structured Query Language. Standardoitu kyselykieli relaatiotietokantoihin tallennettavan datan hallintaan.

XAMPP	X-Apache-MySQL-PHP-Perl-palvelinsovelluspino, jossa X tulee sanasta ”cross-platform”. Suunniteltu toimimaan yhtä hyvin Windowsilla, Linuxilla ja macOS:llä.
AWS	Amazon Web Services. Amazon-yhtiön pilvilaskentapalveluja tarjoava tytäryhtiö.
EC2	Elastic Compute Cloud. AWS:n pilvipalvelu, jossa voidaan konfiguroida ja ottaa käyttöön skaalautuvilla laitteistoresursseilla ja halutulla sovelluspinolla varustettu verkkopalvelininstanssi.
SSH	Secure Shell. Kryptografinen verkkoprotokolla, jota käytetään salatun yhteyden muodostamiseen etätietokoneeseen tai verkkopalvelimeen.
SVN	Apache Subversion. Avoimen lähdekoodin keskitetty versionhallintajärjestelmä.
CSV	Concurrent Versions System. Avoimen lähdekoodin keskitetty versionhallintajärjestelmä.
SHA	Secure Hash Algorithm. Kryptografinen tiivistefunktio, jota käytetään muun muassa salatun verkkoyhteyden muodostamiseen.
DOM	Document Object Model. W3C:n standardoima ohjelmointirajapinta, jossa HTML-, XHTML- tai XML-dokumentti kuvataan puurakenteena, jonka olioita voidaan muokata ohjelmallisesti.
RAIL	Response-Animation-Idle-Load-malli. Toimii verkkosovelluksen käytettävyyden mittarina.
SASS	Syntactically Awesome Style Sheets. CSS-kieltä laajentava skriptikieli, joka käännetään CSS-koodiksi. Yleisimmin käytetty syntaksi on SCSS.
FOUC	Flash of Unstyled Content. Ilmiö, jossa verkkosivu näyttäytyy hetken verkkoselaimen oletustyyleillä, koska selain renderöi sivun sisällön nopeammin kuin tyylitiedostot ehditään ladata.

TinyMCE	Tiny Moxiecode Content Editor. JavaScriptillä kirjoitettu avoimen lähdekoodin selainpohjainen WYSIWYG (what you see is what you get) -editorio-hjain.
IIFE	Immediately Invoked Function Expression. JavaScript-funktio, joka suoritetaan välittömästi sen jälkeen, kun se on määritelty.
WAMP	Windows-Apache-MySQL-PHP/-Perl/-Python-palvelinsovelluspino.
AMI	Amazon Machine Image. Levykuva, joka sisältää tarvittavan informaation virtuaalisen palvelininstanssin varusteluun ja konfigurointiin.
RSA	Rivest–Shamir–Adleman-salausalgoritmi. Epäsymmetrinen salausmetodi, jota käytetään muun muassa verkossa lähetettävän arkaluontoisen datan salaukseen.

1 Johdanto

Yrittäjät ja elinkeinonharjoittajat ymmärtävät verkkonäkyvyyden merkityksen informaatioyhteiskunnassa, jossa Internetin selattavasta osasta, WWW:stä (World Wide Web), on tullut kiinteä osa teollisuusmaan kansalaisen arkielämää. Mikäli yrittäjän toiminimellä ei ole helposti löydettäviä kotisivuja, jotka tarjoavat olennaisen perustiedon yrityksen liiketoiminnasta selkeässä mutta visuaalisesti mielenkiintoisessa muodossa, on tämä lähestulkoon sama kuin yritystä ei olisi olemassakaan. Ei myöskään yksistään riitä, että yrityksellä on olemassa kotisivut. Sivuston on annettava itsestään vaikutelma, kuin se eläisi ja hengittäisi. Toisin sanoen, sivusto sisältää säännöllisin väliajoin päivittyvää sisältöä, sivuston yleinen ulkoasu ja perusilme pyritään pitämään tuoreena vallitseviin verkkoviestinnän trendeihin nähden ja sivusto tarjoaa esteettömästi olennaisen sisällön ja samat perustoiminnallisuudet mahdollisimman monelle eri päätelaitteelle. On selvää, että nämä piirteet yhdessä muodostavat haasteellisen kokonaisuuden, jonka toteutuminen edellyttää hyvien työkalujen ja verkkoalustojen käyttöä.

Verkkosivustot, joiden toteuttamisessa ja ylläpidossa on hyödynnetty sisällönhallintajärjestelmiä, muodostavat nykyään niin suuren osan selattavaa verkkoa, ettei sitä voida jättää huomiotta. Monet sisällön tuottamiseen, tallentamiseen, muokkaamiseen, organisointiin ja julkaisuun liittyvät prosessit ovat näissä sovelluksissa täysin automatisoituja. Lisäksi sisällöntuottajilla on käytössään monenlaisia sivuston sisällön, ulkoasun ja asetusten muokkaamista helpottavia käyttöliittymäkomponentteja. Modernille sisällönhallintajärjestelmälle ominainen piirre on myös se, että se tarjoaa kehyksen sekä useita rajapintoja, joiden avulla kehittäjän on mahdollista laajentaa ja mukauttaa järjestelmän tuomia perustoiminnallisuuksia suhteellisen pienellä vaivalla. Yksinkertaisen ja pienen sivuston käyttöönotto onnistuu WordPressin kaltaisen julkaisujärjestelmän tapauksessa myös teknisesti taitamattomalta henkilöltä, mutta pitkälle räätälöidyn ja ammattimaisen sivuston toteuttaminen edellyttää yhä verkko-ohjelmointitaitoja ja itse järjestelmän teknisen rakenteen syvää tuntemusta.

Opinnäytetyön tavoitteena on perehtyä WordPress-julkaisujärjestelmään verkkokehittämisen ja verkkosovelluskehyksien näkökulmasta sekä useisiin menettelytapoihin, jotka katsotaan sekä yleisesti verkkokehittämisen että WordPress-kehittämisen näkökulmista hyviksi käytänteiksi. Opinnäytetyön raportin tarkoitus ei ole toimia ohjekirjana WordPress-sivuston tai -teeman asentamiseen, koska tätä varten on olemassa

WordPressin omat dokumentaatiot WordPress Codex -hakemistossa (https://codex.wordpress.org/). Insinöörityössä myös suunniteltiin ja toteutettiin helsinkiläisen mainostoimiston WordPress-pohjaisten kotisivujen uudistettu ulkoasu rakentamalla sivustolle räätälöity WordPress-teema. Sivuston ylläpidon sekä sisällön ja ulkoasun muokkaamisen helppous tuottaa erityisen paljon arvoa asiakkaan näkökulmasta, ja siten ne katsottiin keskeiseksi tavoitteeksi projektissa.

2 Sisällönhallintajärjestelmät

2.1 Verkkojulkaisemisen tausta

Internet-aikakauden tulon myötä yhä suurempi osa yritysten datasta ja informaatiosta sijoittuu verkkoon. Koska nopeat verkkoyhteydet ovat nykyään länsimaisessa kulttuurissa tavallinen yleishyödyke suuressa osassa kotitalouksia, Internetistä on luonnollisesti tullut yrityksille ja yrittäjille keskeinen markkinointiviestinnän kanava. Verkkoviestintä ei tässä kontekstissa rajaudu ainoastaan kuluttajaviestintään, sillä yhä enenevissä määrin erikokoiset yritykset hyödyntävät verkkosovelluksia lähes kaikessa viestinnällisessä toiminnassaan. Tähän lukeutuvat muun muassa yrityksen sisäinen tiedonjako, dokumentaation hallinta, toiminnanohjaus, asiakkuudenhallinta, yritys- ja sidosryhmämarkkinointi sekä projektinhallinta.

Perinteiset tavat julkaista ja hallita sisältöä verkossa eivät enää riitä vastaamaan nykyajan tarpeisiin; verkkomedian ja -tiedon kuluttajat odottavat verkkosivuilla tärkeän tiedon olevan välittömästi saatavilla ja sisältöä päivittävän ja ylläpidettävän säännöllisin väliajoin. Luonnollisesti verkkosisällön tulee olla myös helposti löydettävissä sekä ihmisille että hakukoneille, ja sivuston sisäisten linkitysten tulee päivittyä automaattisesti sisällön muuttuessa. Verkkosisällön tuottajat ja graafiset suunnittelijat odottavat verkossa julkaistavan sisällön tuottamisen ja sivuston visuaalisen ilmeen muokkaamisen olevan helppoa ja intuitiivista. Näiden lisäksi brändi-imagon ja ammattimaisuuden esille tuominen edellyttää yrityksen sivuston ilmeeltä ja rakenteelta johdonmukaisuutta riippumatta julkaistavan sisällön formaatista tai määrästä.

Puhuttaessa verkkoviestinnän perinteisestä mallista tässä raportissa viitataan erityisesti vuodesta 1995 aina 2000-luvun alkupuolelle yleisesti käytettyihin toimintatapoihin. Van-

han toimintamallin mukaan pääasiallisin vastuu yrityksen verkkosivujen sisällön muuttamisesta ja päivittämisestä kuului joko yrityksen omalle tietotekniikkaosastolle tai ulkoiselle webmasterille. Sivuston sisällön muuttaminen tehtiin yleensä muodostamalla FTP-yhteys verkkopalvelimelle ja kirjoittamalla sisältö suoraan HTML-tiedostoihin käyttäen tekstieditoria. Käytännössä aina, kun sisältöä tuottava toimisto halusi muokata verkkosivuillaan julkaistavaa sisältöä, sen tuli ottaa ensin yhteys webmasteriin tai muuhun tekniseen tukihenkilöön, joka suoritti sisällön päivittämisen sivustolle edellä mainitulla tavalla. [1, s. 3–4.]

Nykyajan tarpeiden valossa on selvää, että edellä esitelty toimintamalli sisältää puutteita eikä se ole sellaisenaan riittävä vastaamaan yritysten verkkosisällön tuottamisen ja päivittämisen vaatimuksiin. Menettelytapa ei ole pelkästään hidas, vaan se on lisäksi altis virheille; sisällöntuottajat eivät ainoastaan ole voimakkaasti riippuvaisia teknisestä tuesta koko prosessissa, vaan tämän lisäksi itse sisältöä niin sanotusti ”kovakoodataan” käsin suoraan tiedostoihin.

Koska tässä toimintamallissa sisältöä ei juurikaan erotella teknisestä toteutuksesta, sivuston koodin rikkomisen riski sisältöä muokatessa on suuri. Editoijan on myös otettava sisältöä muokatessa huomioon metatiedon ja linkitysten eheys. Kun suurelle, useita kymmeniä tai jopa satoja tiedostoja sisältävälle sivustolle tulee tehdä sisällön rakenteellisia muutoksia, prosessissa tulee manuaalisesti päivittää suuri määrä sivuston eri osien välisiä linkkejä ja yksittäisten sisältöelementtien metatietoja. Tämä hidastaa entisestään päivitystyötä, ja ylipäänsä tällä tavoin toteutettua sivustoa on erittäin työlästä ylläpitää. Suurempien yritysten kohdalla verkkosisällön hallintaan liittyvien ongelmien määrä edelleen kasvaa. Esimerkkinä mainittakoon yritys, jonka sivusto sisältää tietoa ja materiaaleja useilta yrityksen sisäisiltä osastoilta, joiden tulee vastata näiden materiaalien ajanmukaisuudesta ja oikeellisuudesta. Useiden eri ihmisten tai tahojen tehdessä muutoksia sivuston eri osiin tarve muutosten seurannalle ja sivuston elinkaaren eri versioiden hallinnalle kasvaa. [1, s. 3–4; 2.]

Lähestyttäessä 2000-lukua staattisen verkkosisällön tuottaminen, muokkaaminen ja ylläpito huomattiin pian toimimattomaksi ja vanhentuneeksi ratkaisuksi. Tässä yhteydessä staattisuudella tarkoitetaan hyvin pitkälle sitä, että verkkosisällön tuottaminen ja päivittäminen olivat hyvinkin manuaalisia prosesseja ja sisältö oli täysin ennalta määriteltyä, ei siis millään tapaa kohdennettua tai käyttäjäkohtaisesti räätälöityä.

Markkinoilla yleistyi useita erilaisia palvelinpään ohjelmointikieliä ja tekniikoita, jotka kykenivät generoimaan HTML-sisältöä perustuen haluttuihin syötteisiin. Tunnetuimmista dynaamisten verkkosivujen luontiin käytetyistä tekniikoista mainittakoon PHP- ja Perl-komentosarjakielet sekä Java-ohjelmointikielen Servlet- ja JavaServer Page -tekniikat. Näistä tekniikoista PHP saavutti web-kehittäjien ja -harrastelijoiden keskuudessa suuren suosion suhteellisen yksinkertaisen syntaktinsa ja matalan oppimiskäyrän ansiosta. Muita PHP:lla toteutettujen verkkosivustojen ja -sovellusten leviämiseen vaikuttavia tekijöitä ovat muun muassa kielen avoin lähdekoodi, ilmaisuus, tuki useille eri palvelinympäristöille ja monipuolinen joukko sisäänrakennettuja komponentteja ja kirjastoja. [3; 4.]

Erilaisilla palvelinpään tekniikoilla mahdollistettiin verkkosisällön dynaaminen koostaminen. Sivut voitiin rakentaa useista eri koodipalikoista, komponenteista tai proseduureista. Sisältö saatiin enemmän erotetuksi esittämistavasta käyttämällä sivumalleja, joihin tulostettava data haettiin erillisestä lähteestä, kuten relaatiotietokannasta. Mahdolliseksi tuli myös muun muassa evästeiden ja istuntojen käyttö sekä käyttäjän tekemien peräkkäisten tai toistuvien klikkauksien seuranta. Näin verkkosivustoista saatiin tehokkaampia, käyttäjäkohtaisesti räätälöityjä sekä huomattavasti helpompia ylläpitää ja laajentaa. [5, s. 58–59.]

2.2 Sisällönhallintajärjestelmät verkkokehityksessä

Palvelinpään ohjelmointikielten kehittymisen ja laajemman käyttöönoton yhteydessä kehittyivät myös kullekin kielelle ominaiset työkalut verkkosivustojen ja -sovellusten rakentamiseen. Ohjelmakirjastoista ja muista komponenteista koostettiin yhä laajempia kokonaisuuksia, jotka tunnetaan yleisesti verkkosovelluskehyksinä. Verkkosovelluskehyksillä pyritään yksinkertaistamaan monia verkkosovelluskehityksessä toistuvia toimenpiteitä, kuten HTTP-pyyntöjen reitittäminen tiettyihin ohjainkomponentteihin, evästeiden ja istuntojen hallinnoiminen, tietokantayhteyksien muodostaminen sekä asiakaspään näkymien renderöintiin käytettävien sivupohjien rakentaminen. Kaksi merkittävää varhaista verkkosovelluskehystä ovat Microsoftin ASP ja Sun Microsystemsin (nykyisin Oracle) JavaEE. [6; 7.]

Vaikka palvelinpään tekniikoiden kehittyminen ja erilaisten sovelluskehyksien käyttö tehostivat huomattavasti verkkosivustojen kehitystä, sivustojen rakentaminen ja ylläpitä-

minen edellyttivät syvää tietoteknistä osaamista, ohjelmointitaitoja ja verkkoharrastuneisuutta. Uuden verkkosisällön julkaiseminen tai vanhan päivittäminen oli vuosituhannen vaihteessa edelleen monelle vähemmän tekniselle henkilölle haastavaa. Kilpailu eri selainohjelmien, tunnetuimpina Netscape ja Internet Explorer, välillä mahdollisti yhä monimuotoisemman HTML-merkinnän rakenteen ja kehittyneempien teknologioiden käytön verkkosivustoilla, ja näin ollen tiedon esittämisen laatutaso verkossa parani. Tätä myötä, luonnollisesti, verkkoselailu herätti nopeasti yhä suuremman yleisön kiinnostuksen ja tunnettujen yritysten sekä valtion laitosten sivustoilta odotettiin usein päivittyvää sisältöä. Sovelluskehittäjät ja tietotekniikkahenkilöt toimivat kuitenkin edelleen sisällöntuottajien välikätenä verkkojulkaisemisessa, mikä tunnistettiin merkittäväksi pullonkaulaksi prosessissa. [8, s. 31; 9, s. 64.]

Verkkokehityksessä huomattiin tarve kokonaisvaltaiselle järjestelmälle, joka yhä paremmin erottaisi verkkosivuston teknisen rakenteen ja toiminnallisuudet julkaistavasta sisällöstä. Järjestelmään tulisi olla mahdollista integroida sisällön sekä sisällön rakenteen ja esitystavan muokkaamista helpottavia käyttöliittymäkomponentteja, joiden käyttö ei edellytä ohjelmointitaitoja tai tietoteknistä osaamista. Keskeiseksi vaatimukseksi katsottiin myös sisällön indeksoitavuus ja mahdollisuus hallita metatietoa määrittelemällä muun muassa olioiden keskinäisiä suhteita hierarkioilla, luokitteluilla, avainsanoilla tai nimikkeillä.

Se, mikä mielletään yrityksen verkkoviestinnän näkökulmasta sisällöksi, vaihtelee tapauskohtaisesti. Tähän luonnollisesti vaikuttaa julkaisijan toimiala sekä viestinnälliset ja liiketoiminnalliset päämäärät. Verkkosisällönhallinnassa sisältöä voidaan kuitenkin karkeasti luonnehtia sisällyttämällä siihen vähintään seuraavat oliot:

- uutiset, tapahtumat, kalenterit, blogiartikkelit, käyttäjäkommentit ja staattiset sivut
- staattiset tekstidokumentit ja liitteet
- lyhytaikaiset digitaaliset aineistot tai hyödykkeet, kuten liput, kupongit ja tiedotteet
- sivustolle ladattavat multimediatiedostot, kuten kuvat, äänitallenteet, videot tai mikä tahansa informaatiokappale, jonka esittämiseen tarvitaan erillinen ohjelma tai liitännäinen
- mikä tahansa uudelleenkäytettävä data, joka varastoidaan tietokantaan. [10.]

Sisällönhallintajärjestelmä on käsitteenä varsin tulkinnanvarainen: sillä voidaan viitata moniin erilaisiin järjestelmiin, joissa päämääränä on suunnitelmallisesti organisoida, jakaa, editoida, arkistoida ja julkaista yrityksen digitaalisia aineistoja ja resursseja. Tässä raportissa käsite rajataan yleishyödyllisiin verkkosisällönhallintajärjestelmiin, jotka soveltuvat pienten ja keskisuurten yritysten sekä yksityisten elinkeinonharjoittajien kotisivujen suunnitteluun, kehittämiseen ja ylläpitoon. PHP-kielellä rakennetuista avoimen lähdekoodin sisällönhallintajärjestelmistä WordPress, Joomla ja Drupal ovat edelleen varteenotettavimpia vaihtoehtoja ammattimaisten verkkosivustojen kehitykseen. W3Techs.com-sivuston keräämän teknologiatrendidatan mukaan 30 % maailman verkkosivustoista käyttää WordPressiä vuonna 2018 [11].

Väistämättä yksi tekijä WordPressin jatkuvasti kasvavaan käyttöönottoon on lähdekoodin avoimuus. Avoin lähdekoodi on hajautetun tuotannon malli, jossa projektin tuotteet, kuten kehitettävän sovelluksen lähdekoodi, pohjapiirustukset ja dokumentaatio, ovat julkisesti saatavilla vastikkeettomasti lisenssillä, joka kattaa oikeudet tutkia, modifioida ja jakaa sovellusta kenelle tahansa ja mihin tarpeisiin tahansa [12]. Avoimen lähdekoodin tuoma julkisuus ja läpinäkyvyys taas puolestaan edesauttaa projektin vakaan ja aktiivisen kehittäjäyhteisön nopeaa kasvua. Yhä enenevissä määrin uusien sovelluskehitysprojektien tuotteita julkaistaan avoimena lähdekoodina, ja tämä on useimmiten todettu menestyksekkääksi tuotantomalliksi. On kuitenkin todennäköistä, että mahdollisesti tuhansista intohimoisista kehittäjistä koostuvan joukon ilmaiseksi tuoma osallistuminen tuottaa laadukkaampaa sovelluskoodia ja innovatiivisia ratkaisuja nopeammin kuin paljon pienemmän ja suljetun ryhmän työpanos.

Muita merkittäviä tekijöitä WordPressin suosion takana ovat alustan käyttöönoton helppous sekä joustavuus ja laajennettavuus. Lisäosia on saatavilla WordPressin virallisesta lisäosahakemistosta yli 50 000, joista monien peruskäyttö ei edellytä minkäänlaisia ohjelmointitaitoja. Lukuisien tunnettujen lisäosien kehittäjät soveltavat niin kutsuttua freemium-ansaintamallia, jossa sovelluksen peruskäyttöön tarkoitetut ominaisuudet ovat saatavilla ilmaiseksi, kun taas erilaisia edistyneempiä lisätoiminnallisuuksia tarjotaan sisäisen hinnoittelukäytännön mukaisia maksuja vastaan. Yksittäisen sivustolle asennetun lisäosan rooli on yleensä vastata yhteen selkeästi rajattuun ja pieneen ongelmaan, esimerkiksi palautelomakkeen konfigurointi ja esittäminen sivupohjassa. Toisaalta lisäosa voi myös olla sivuston tavanomaisen toiminnan kannalta niin keskeinen ja kokonaisval-

tainen, ettei sivusto voisi toimia ilman kyseistä lisäosaa. Tästä hyvänä esimerkkinä mainittakoon E-commerce-sivusto, jonka verkkokaupan toiminta nojautuu Woocommerce-lisäosaan ja sen oheistuotteisiin.

Koska WordPress-ytimen ja lisäosien kehityksen taustalla on aktiivinen kehittäjäyhteisö, jossa kannustetaan avoimeen keskusteluun lähdekoodiin liittyvistä ongelmista ja ohjelmointivirheistä, ohjelmistopäivityksiä julkaistaan suosittuihin komponentteihin suhteellisen usein. Tähän liittyy toisaalta myös ongelmia sivustojen ylläpidettävyyden osalta. Tietyn lisäosan päivittämisen myötä muuttunut sovelluksen palanen saattaa päätyä ennalta-arvaamattomaan konfliktiin toisen lisäosan ohjelmakoodin kanssa. Myös tapaukset, joissa lisäosan päivityksessä käyttöönotetut muutokset rikkovat tavalla tai toisella käytössä olevan teeman ulkoasun, ovat yleisiä. Sivuston ylläpitäjän kuitenkin tulee huolehtia, että WordPress ja sen käyttämät lisäosat ovat sivustolla jatkuvasti päivitettyinä uusimpiin versioihin, etteivät vanhentuneet komponentit altistaisi sivustoa tietoturvauhkille.

Väistämättä WordPressin kaltaisen suositun julkaisujärjestelmän käyttö verkkosivustojen toteutuksessa kasvattaa huomattavasti todennäköisyyttä, että sivusto päätyy ennemmin tai myöhemmin tietoturvahyökkäyksen kohteeksi. Tietoturvayritys Sucuri julkaisi vuonna 2016 raportin tutkimuksesta, jossa analysoitiin yli 8 000 tietoturvahyökkäyksien ja haittaohjelmien kohteeksi päätynyttä, julkaisujärjestelmän päällä toimivaa sivustoa. Näistä sivustoista selkeästi suurin osa, 74 %, käytti WordPressiä. Murretuista sivustoista 61 % käytti vanhentunutta WordPress-asennusta. Hakkeroitujen WordPress-sivustojen korkea lukumäärä ei korreloi itse WordPress-ytimen tietoturvan kanssa, eikä se tarkoita sitä, että WordPress itsessään olisi vähemmän tietoturvallisempi muihin sisällönhallintajärjestelmiin verrattuna. On huomattavasti yleisempää, että tietoturvaongelmat aiheutuvat erilaisista huolimattomuuksista liittyen sivuston tuotantoon vientiin, palvelinympäristön konfigurointiin sekä sivuston kokonaisvaltaiseen ylläpitoon. [13.]

3 WordPress verkkokehitysalustana

3.1 WordPress verkkosovelluskehyksenä

WordPress oli alun perin vain yksinkertainen alusta blogien kirjoittamiseen ja julkaisuun. Tarkalleen ottaen, WordPress oli ohjelmahaarautuma vuonna 2001 julkaistusta, yksinkertaisesta blogien ja uutisten julkaisutyökalusta b2/cafelog. Lähtökohta molemmissa

ohjelmissa oli kutakuinkin sama: mahdollistaa dynaamisesti tuotettujen blogisivujen julkaiseminen käyttämällä kahta suosittua avoimen lähdekoodin palvelinpään teknologiaa, PHP-komentosarjakieltä ja MySQL-relaatiotietokantaohjelmistoa. WordPressin toiminnan perusperiaate on edelleenkin hyvin pitkälle sama: tallentaa editointinäkymän kautta syötetty data MySQL-tietokantaan ja tuottaa PHP-ohjelmakoodilla sovelluksen näkymä tulostamalla tietokantatauluihin tallennettu data HTML-sivupohjiin. [14, s. 3.]

Tämän raportin kirjoittamishetkellä on kuitenkin turvallista olettaa, että WordPress on ollut jo reilun tovin paljon muutakin kuin vain blogialusta. Tähän asti alustaan integroitujen komponenttien, huomattavan laajennettavuuden sekä aktiivisen käyttäjä- ja kehittäjäekosysteemin puitteissa voidaan jo puhua varsin vakaasta ja varttuneesta sisällönhallintajärjestelmästä. Tämän puolesta puhuu lisäksi se tosiasia, että alustaa käytetään yhä enenevissä määrin muihin tarkoituksiin kuin perinteiseen blogaamiseen, kuten yritysten kotisivujen, verkkokauppojen, tuotesivustojen, portfolioiden, kuvagallerioiden ja videoblogien tuotantoon.

Asiakkaalle tehtävässä verkkokehitysprojektissa verkkokehittäjälle varsin luonteva ratkaisu on valita sivuston tai sovelluksen tuotantoon yksi tai useampia sovelluskehyksiä sen perusteella, mitkä tekniset ja toiminnalliset vaatimukset tuotteelle kussakin kontekstissa asetetaan. Yksi keskeisimpiä motiiveja käyttää sovelluskehyksiä verkkokehityksessä on hyödyntää sellaisia uudelleenkäytettäviä ohjelmamoduuleja ja funktioita, jotka tarjoavat verkkosovelluksissa usein toistuvia toimenpiteitä. Eräs tärkeä esimerkki tällaisista moduuleista ovat erilaisia tietoturvatoimintoja, kuten käyttäjäautentikointia ja syötedatan validointia, tarjoavat ohjelmapaketit ja funktiot. Hyvän sovelluskehyksen piirteenä on olennaisesti myös se, että se tarjoaa verkkokehittäjille perustan helposti kirjoittaa räätälöityä toiminnallisuutta ja logiikkaa projektin tarpeiden puitteissa. [15, s. 31–32.]

Verkkosivuston tai -sovelluksen kehitykseen on tavallista valita sellainen ohjelmointikehys, joka tukee niin kutsuttua **MVC** (**Model**, **View** & **Controller**) -arkkitehtuurimallia. Model, eli malli, koteloi sovelluksen perustoiminnallisuuden, datan ja tietokantayhteydet olioihin. View, eli näkymä, taas edustaa kaikkea käyttäjälle esitettävää informaatiota. Yleensä kaikki sovelluksen tulostama HTML-merkintä ja käyttöliittymät sisällytetään sovelluksen näkymäkerrokseen. Lopuksi Controller, eli ohjain, toimii sovelluksen koordinaattorina, jonka tehtävänä on vastaanottaa asiakasohjelman pyynnöt, käsitellä mallikerroksen olioita, yhdistää olioiden sisältämä data näkymään ja palauttaa tämä näkymä asiakkaalle. Se, millä perusteella ohjain valitsee tietyn datan ja näkymän palautettavaksi,

riippuu siis vastaanotetusta pyynnöstä ja ohjelmoidusta logiikasta. Lähtökohtana sovelluksen toiminnan jaottelulla näihin kolmeen kerrokseen on selkeyttää verkkokehitystä ja tehdä sovelluskoodista ylläpidettävämpää ja modulaarisempaa. [16.]

WordPress ei sellaisenaan tue MVC-mallia. Sen sisäisen toiminnan katsotaan sen sijaan mukailevan läheisesti erästä toista ohjelmistoarkkitehtuurimallia, tapahtumakeskeistä mallia (engl. event-driven model). Vaikka tällainen sovellusarkkitehtuuri ei varsinaisesti vastaa mitään ohjelmoinnin suunnittelumallia, periaatteeltaan se muistuttaa teoksessa **Design Patterns: Elements of Reusable Object-Oriented Software** (1994) esiteltyä tarkkailijamallia (engl. Observer pattern). Tekijäjoukko (niin kutsuttu ”Gang of Four” eli Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides) määrittelee mallin seuraavalla tavalla:

> Olioiden välille määritellään one-to-many-riippuvuussuhde siten, että yhden olion tilan muutoksesta tiedotetaan kaikille riippuvaisille olioille ja nämä päivittyvät automaattisesti.

Malli tunnetaan myös nimellä julkaisija-tilaajamalli (engl. publisher-subscriber pattern). Sekä tarkkailijamallissa että tapahtumakeskeisessä arkkitehtuurimallissa ohjelmiston toiminta tapahtuu varsin suoraviivaisella tavalla: sovelluksen julkaisijakomponentti lähettää tiedonannon tapahtumasta kaikille niille tilaajakomponenteille, jotka on kytketty tai rekisteröity kuuntelemaan kyseistä tapahtumaa. Kukin tilaajakomponentti puolestaan taas reagoi tähän tapahtumaan suorittamalla sille ohjelmoitua logiikkaa. Tapahtumakeskeisyyttä noudattaa hyvin laaja kirjo erilaisia ohjelmistotuotannon, verkon ja tietotekniikan sovelluksia. Esimerkkinä näistä mainittakoon selaimessa suoritettava JavaScript-koodi, jossa sivun dokumenttioliomallia käsittelevä funktio on kytketty hiiren klikkausta tarkkailevaan tapahtumakuuntelijaan. [17; 18.]

Koska WordPressin ytimen toiminta noudattaa tapahtumakeskeistä sovellusarkkitehtuuria, verkkokehittäjillä on mahdollisuus liittää räätälöityä toiminnallisuutta tiettyihin WordPressin ajon aikaisiin tapahtumiin. Tällaiset tapahtumat voivat liittyä sivulatauksen elinkaaren eri vaiheisiin, tietokannasta lukuun ja sinne kirjoittamiseen, artikkelin sisältödatan tulostukseen sivumalliin ja moniin muihin ennalta määrättyihin toimenpiteisiin. WordPress-teeman ja -lisäosien kehittäjillä on mahdollisuus luoda helposti myös omia tapahtumia, joihin muut kehittäjät voivat edelleen kytkeä täysin omaa toiminnallisuutta ja tehdä näin komponenteista erittäin modulaarisia ja laajennettavia. WordPressin tapahtumakeskeisessä arkkitehtuurissa lisäosarajapinta (engl. Plugin API) ja sen tarjoamat

toiminta- ja suodatinkoukut (engl. action & filter hooks) toimivat tärkeässä roolissa. Niitä käsitellään tarkemmin raportin luvussa 3.3.

WordPressin voimakkaasti kasvavaan käyttöönottoon ja suosioon amatöörikoodarien keskuudessa on väistämättä vaikuttanut se tosiasia, että luonteeltaan lineaarinen tapahtumakeskeinen arkkitehtuuri koetaan yksinkertaisemmaksi ymmärtää verrattuna MVC-malliin. Toisaalta tämä on myös tuottanut monille MVC-malliin pohjautuvia sovelluskehyksiä käyttäville kehittäjille vahvan mielihalun välttää käyttämästä WordPressiä verkkosovellusten kehittämisessä. WordPressin päälle rakennetuissa sovelluksissa on hyvin tavallista nähdä koodia, jossa MVC-mallin ohjain- ja mallikerrosten logiikka sekoittuu keskenään ja hajaantuu sovelluksen eri tiedostoihin. WordPress-ytimen rakenteesta on ajan myötä kehittynyt myös proseduraalisen ohjelmoinnin ja olio-ohjelmoinnin paradigmojen yhdistelmä. Tämä selittyy osittain luonnollisesti sillä, että WordPressin ensimmäiset kehityshaarat mukautuivat voimakkaasti proseduraaliseen ohjelmointimalliin. [15, s. 31–32 ja 46.]

WordPressin tapahtumakeskeinen arkkitehtuuri ja juuret proseduraalisessa ohjelmoinnissa eivät kuitenkaan tarkoita sitä, että kehittäjän ei olisi mahdollista kirjoittaa omaa teemaa olio-ohjelmoinnin pohjalta tai esimerkiksi lisäosaa, joka mukauttaa WordPressin sisäistä toimintaa muistuttamaan enemmän MVC-arkkitehtuuria. On kuitenkin eri asia, onko näin perustelua tehdä tai tuoko se jotain erityistä lisäarvoa projektin laadun tai asiakkaan tarpeiden näkökulmasta. Verkkosivuston tai -sovelluksen ammattimaisuus ei kuitenkaan korreloi suoraan sen kanssa, rakennetaanko sovellus käyttämällä esimerkiksi MVC-verkkokehystä tapahtumakeskeisen julkaisujärjestelmän sijaan. Projektin lopputuotteen laatuun vaikuttaa merkittävästi sopivan verkkokehitysalustan valinta huomioiden projektikohtaiset tarpeet ja tekniset vaatimukset sekä se, kuinka laajasti alustan tarjoamia toiminnallisuuksia, metodeita ja käytäntöjä sovelletaan kunkin ongelman ratkaisuun.

3.2 Teema ja sivupohjat

Teema toimii keskeisenä osana WordPress-sivuston ulkoasun ja asiakaspään toiminnallisuuden suunnittelussa. Käytännössä teema on verkkosivuston ulkoisen rakenteen pohja-asetelma, joka koostuu sivupohjatiedostoista (engl. template file). Teema ei ole

välttämättä rajoittunut pelkästään sivuston ulkoasuun, sillä kehittäjällä on lisäksi mahdollisuus rakentaa mielivaltainen määrä erilaisia palvelinpään toiminnallisuuksia teemakohtaisesti mihin tahansa PHP-tiedostoon. WordPress-teemoja ei tuotteisteta pelkästään yleiskäyttöisiksi sivustopohjiksi muokattavine tyyleineen, vaan on myös varsin tavallista nähdä teemoja, joiden ulkoasu ja toiminnallisuudet on rakennettu tilaustyönä yksinomaan kyseisen sivuston omistajalle. Teemaa voidaan siis karkeasti kuvailla yksittäisen sivuston runkona ei pelkästään ulkoasun vaan myös yleisten toiminnallisuuksien osalta.

Kukin teema asetetaan omaan kansioon WordPressin teema-polun sisälle, oletuksena wp-content/themes/. Itse sivupohjatiedostojen lisäksi teema sisältää kaikki sille ominaiset tyyli- ja JavaScript-tiedostot, vaihtoehtoisen functions.php-tiedoston sekä verkkoaineistoa, kuten kuvia, grafiikkaa tai muita mediatiedostoja. Suurin osa kaikesta teemaa koskevasta tyylittelystä sisällytetään styles.css-tiedostoon, joka on index.php-sivupohjatiedoston lisäksi ainoa pakollinen tiedosto WordPress-teeman sisällä. [19.]

Sivupohjat ovat monissa teemoissa sivuston tärkeimpiä rakennuspalikoita. Se, miten sivupohjat on rakennettu ja miten ne yhdistetään tietyissä tilanteissa, määrittelee kunkin käyttäjälle palautetun lopullisen sivun rakenteen. Sivupohjat ovat luonteeltaan modulaarisia ja uudelleenkäytettäviä; teeman kaikki sivut voivat sisältää kaikki seuraavat kolme yleiskäyttöistä peruspohjaa:

- Header.php eli sivun otsakemalli sisältää kaiken tavanomaisen, verkkosivun alkuosassa esitetyn HTML-merkinnän, kuten DOCKTYPE-määritteen, avaavan html-tagin, head-osion, avaavan body-tagin, otsake-elementin HTML-merkinnän ja niin edelleen.
- Sidebar.php yleensä sisältää dynaamisia alueita joko teeman tai eri lisäosien vimpaimille (pienoisohjelma, engl. widget). Huomioitavaa on, että sivupalkki voidaan nimestään huolimatta asettaa mihin kohtaa sivua tahansa.
- Footer.php sisältää kaiken HTML-merkinnän, jota tavallisesti nähdään sivun loppuosassa, kuten sivuston alapalkin HTML-merkintä, sulkevat body- ja html-tagit ja niin edelleen.

Otsake- ja footer-sivupohjien kohdalla tulee huolehtia, että sivupohjiin sisällytetään WordPressin funktiot wp_head ja wp_footer, edellinen ennen sulkevaa head-tagia ja jälkimmäinen ennen sulkevaa body-tagia. Tämä on tärkeää, koska teema ja lisäosat käyttävät näitä funktioita JavaScript- ja tyylitiedostojen liittämiseen sivulle. [20, s. 20–24.]

Vaikka kehittäjän olisi toki mahdollista liittää edellä mainitut peruspohjat sivun runkoon käyttämällä tavanomaista PHP-kielen include-funktiota, WordPress tarjoaa tähän tarkoitukseen erikoistuneet funktiot get_header, get_sidebar ja get_footer. Näistä funktioista tekevät erityisen joustavia niiden vastaanottamat merkkijonoparametrit, joilla voidaan kutsua kutakin sivupohjaa nimellä, esimerkiksi get_sidebar('secondary'). Tällöin WordPress etsii sivupohjan, jonka tiedostonimessä esiintyy parametrina annettu merkkijono, ja liittää sen siihen sivupohjaan, jossa funktiota alun perin kutsuttiin. Tämä on erityisen hyödyllistä, kun halutaan sivukohtaisesti tai muutoin tilanteesta riippuen käyttää erilaista rakennetta otsakkeelle, sivupalkille tai footerille. [20, s. 24.]

WP-luokan parse_request-metodilla ja WordPressin mallihierarkialla on tärkeä rooli, kun WordPressin tulee päättää, mikä yksittäinen sivupohja tai joukko sivupohjia palautetaan kunkin HTTP-pyynnön kohdalla. Ensin parse_request-funktion tulee poimia URL-osoitteen kyselyosa ja itse sivuston kotiosoite. Seuraavaksi funktio noutaa kestolinkkien uudelleenkirjoitussäännöt ja iteroi sääntöjen joukon läpi, kunnes se löytää kestolinkkiä vastaavan säännön. Osuman löytyessä funktio vastaanottaa voimassa olevat kyselymuuttujat säännössä määritellystä informaatiosta. Mikäli osumaa ei löydy, vastauksena palautetaan 404-virhesivu. Jokaisen kyselymuuttujan kohdalla tarkistetaan, onko se asetettu aktiiviseksi kestolinkkien jäsentelyssä, HTTP GET -pyynnössä tai HTTP POST -pyynnössä. Kaikki aktiiviset kyselymuuttujat tallennetaan assosiatiiviseen taulukkoon query_vars. [21.]

Lyhyesti ilmaistuna, parse_request-funktion tehtävä on jäsentää URL-osoite kyselymuuttujataulukoksi, joka välitetään edelleen WP_Query-luokan instanssille. WP_Query-luokka on WP-luokan ohella WordPressin normaalin toiminnan kannalta tärkeä komponentti, sillä sen tulee määritellä, minkä tyypin sisältöä HTTP-kysely koskee, suorittaa asianmukaiset tietokantakyselyt ja varastoida kyselyn tuloksena saadut sisältöoliot tulostusta varten myöhemmin WordPress-silmukan sisällä. Kun WordPress on tietoinen pyyntöä koskevan sivun tyypistä, voidaan asianmukaisen sivupohjatiedoston etsintä sivupohjahierarkiasta aloittaa. [21; 22.]

Sivupohjahierarkia on WordPress-teemojen räätälöintiä yksinkertaistava valintajärjestelmä, joka toimii verkkokehittäjän tukena monissa sivuston ulkoasun suunnitteluun liittyvissä ongelmatilanteissa. Esimerkkinä mainittakoon tilanne, jossa asiakas haluaa sivustonsa kahden eri artikkelityypin, tuotteet ja tapahtumat, sivujen noudattavan erilaista

visuaalista rakennetta. Sivupohjan haku aloitetaan mahdollisimman spesifistä tilanteesta, ja vertailua jatketaan hierarkiassa aina yleisluontoisempaan suuntaan. Oletetaan, että käyttäjä hakee yksittäisen artikkelityypin sivua. Tällöin

1. WordPress tarkistaa, löytyykö teeman kansiosta sivupohjatiedostoa, jonka tiedostonimi sisältää pyydetyn artikkelin tyypin ja polkutunnuksen single-{artikkelityyppi}-{polkutunnus}.php

2. jos ei, etsitään pyydetyn artikkelityypin sivupohjatiedosto single-{artikkelityyppi}.php

3. jos ei, hierarkian polkua jatketaan edelleen tiedostonimeen single.php ja siitä edelleen singular.php

4. mikäli yhtäkään aikaisemmissa vaiheissa mainittua tiedostoa ei löydetä, WordPress palauttaa lopullisen oletussivupohjan index.php. [23.]

WordPressin sivupohjahierarkia tukee saumattomasti lapsiteemojen käyttöä. Lapsiteemat tarjoavat kehittäjälle mahdollisuuden periä kaiken toiminnallisuuden kolmannen osapuolen teemasta muokkaamatta itse varsinaisen teeman koodia. Luomalla lapsiteeman kansioon sivupohjatiedosto korvataan samanniminen tiedosto isäntäteeman kansiosta. Mikäli edellä mainitussa esimerkkitapauksessa kehittäjä käyttäisi omaa lapsiteemaa, sivupohjahierarkia tarkistaa ensimmäisenä, sisältääkö lapsiteema sivupohjaa tiedostonimellä single-{artikkelityyppi}-{polkutunnus}.php. Jos sisältää, tämä tiedosto yliajaa mahdollisen isäntäteeman kansiossa sijaitsevan samannimisen sivupohjatiedoston. [24.]

Lapsiteemojen käyttö katsotaan hyväksi käytänteeksi kaikissa niissä tapauksissa, joissa kolmannen osapuolen rakentamaa teemaa tulee tavalla tai toisella muokata omiin tarpeisiin sopivaksi. Mikäli tällaisen teeman tiedostoja ja koodia muokattaisiin suoraan teeman kansion sisällä, on erittäin korkea todennäköisyys, että nämä muutokset hävitetään ja sivuston toiminta rikkoontuu uuden teeman versioon tehtyjen päivitysten myötä.

3.3 Lisäosarajapinta

Koska WordPress on tapahtumakeskeinen sovellus, se tarvitsee mekanismin, jonka avulla ulkopuoliset komponentit voivat kytkeytyä ajon aikana suoritettaviin toimintoihin ja lisätä verkkosivustoa täydentäviä toiminnallisuuksia muokkaamatta WordPressin ydinkomponentteja. Lisäosarajapinta on yksi WordPressin tarjoamista sovellusrajapinnoista, jotka toimivat tälle suositulle julkaisujärjestelmälle tunnusomaisen laajennettavuuden ja joustavuuden taustalla. Miten sovellusrajapinta, tai **API** (**Application Programming Interface**), tarkalleen ottaen määritellään ja mitä se pitää sisällään, vaihtelee kontekstin mukaan. Perusajatus kuitenkin on, että sovelluksella tai sen aliohjelmilla olisi standardisoitu ja dokumentoitu liittymäkohta, jonka avulla kolmansien osapuolten sovellukset voivat pyytää järjestelmää suorittamaan erilaisia yleishyödyllisiä toimenpiteitä tai saumattomasti lisätä mukautettuja toiminnallisuuksia.

Tässä yhteydessä rajapinta koostuu käytännössä joukosta erilaisia PHP-funktioita ja -luokkia, joita kehittäjä voi käyttää rakennuspalikoina laajentaessaan WordPress-sivuston toimintaa omiin tarpeisiin sopivaksi. WordPress-ydin suorittaa sivupyynnön elinkaaren eri vaiheissa eräänlaisia nimettyjä tapahtumia, joita kutsutaan WordPress-kehittäjäyhteisössä koukuiksi, tarkemmin toiminnoiksi ja suodattimiksi. Nämä koukut eivät itsessään suorita mitään logiikkaa, vaan toimivat enemmänkin paikanpitäjinä teemassa tai lisäosassa määritetyille funktioille, jotka kutsutaan koukun laukaisuvaiheessa. Nimeämiskäytäntö on hieman harhaanjohtava, sillä kehittäjäyhteisössä on tapana viitata toiminnoilla ja suodattimilla sekä koukkuihin että koukkuihin kytkettyihin funktioihin. Selkeyden vuoksi on kuitenkin syytä painottaa, että koukuista puhuttaessa käytetään termejä toiminta- ja suodatinkoukut ja funktioiden tapauksessa toiminta- ja suodatinfunktiot.

Toimintakoukut

Toimintakoukut (engl. Action hook) ovat paikanpitäjiä sellaisille teeman tai lisäosan kehittäjän kirjoittamille funktioille, joiden halutaan suorittavan tietyn diskreetin toimenpiteen tiettynä ajankohtana. Toimintakoukkujen tarkoitus on joko lisätä tai poistaa mukautettua koodia. Tavallisia toimenpiteitä, jotka toteutetaan käyttämällä toimintakoukkuja, ovat esimerkiksi JavaScript-tiedostojen lisääminen etusivun head-osioon tai käyttäjän uudelleenohjaus mukautettuun malliin tietyn sivupyynnön kohdalla. Itse funktion koukkuun kytkentä koostuu kahdesta komponentista. Toimintafunktioiden kytkentään käytetään add_action-funktiota seuraavasti:

```
add_action( string $koukun_nimi, callable $lisättävä_funktio, int $priori-
teetti, int $argumentit );
```

Funktion kolmas parametri ilmaisee toimintafunktion tärkeysasteen. Pienemmän kokonaislukuarvon funktio asetetaan tärkeysjärjestyksessä korkeamman lukuarvon funktioiden edelle. Tämä tarkoittaa, että kyseisen koukun kohdalla korkeamman prioriteetin funktio suoritetaan ennen muita. Mikäli kahdella funktiolla on sama prioriteettiarvo, funktiot suoritetaan siinä järjestyksessä, missä ne lisättiin toimintakoukkuun. Neljäs parametri on kokonaislukuarvo, joka ilmaisee toimintafunktion vastaanottamien parametrien lukumäärän. Toimintafunktiot eivät palauta mitään arvoja tai dataa. [25.]

Toimintakoukkuun kytkennän seuraava komponentti on itse toimintafunktion määrittely. Havainnollistamisen vuoksi esimerkkikoodissa 1 on esitetty eräs toimintakoukku ja siihen kytketty funktio, jotka WordPress lisää sivustoon oletuksena.

```
add_action('wp_head', 'noindex', 1);

function noindex() {
    // If the blog is not public, tell robots to go away.
    if ( '0' == get_option('blog_public') )
        wp_no_robots();
}

function wp_no_robots() {
    echo "<meta name='robots' content='noindex,follow' />\n";
}
```

Esimerkkikoodi 1. wp_head-toimintakoukku ja siihen kytketty noindex-funktio [26; 27].

wp_head-toimintakoukkuun liitetyt funktiot suoritetaan sivun head-osion sisällä. Esimerkkikoodin ajatus on yksinkertainen: jos sivustoa ei ole merkitty asetuksista julkiseksi, sivun head-osioon tulostetaan robots-metaelementti, jolla kielletään hakukoneita indeksoimasta kyseistä sivua [26].

Kehittäjällä ei ole käytössään pelkästään laaja valikoima WordPress-ytimeen määriteltyjä toimintakoukkuja, vaan koukkuja voidaan luoda helposti myös itse. Jotta itsetehtyyn koukkuun kytketty funktio voitaisiin suorittaa, tulee toimintafunktion määrittelyn ja koukkuunkytkennän lisäksi käyttää do_action-funktiota. Tämä funktio vastaanottaa parametrina omatekoisen koukun nimen, ja sitä kutsutaan siis siinä sivupohjatiedostossa tai tapahtumaketjussa, jossa sen halutaan suorittavan toimintafunktio. Koukun nimenannossa on syytä noudattaa yleisiä nimeämiskäytäntöjä. Koukun nimen tulisi selkeästi ilmaista,

mihin toimintoon ja ajankohtaan tämä liittyy. On hyvä tapa käyttää nimessä etu- ja loppuliitteitä, kuten ”pre”, ”before”, ”begin”, ”after” ja ”end”, joilla ilmaistaan toiminnan ajankohta suhteessa tiettyyn tapahtumaan. Tämän lisäksi teeman ja lisäosien kehittäjien tulee kirjoittaa koukkujen ja funktioiden nimeen etuliitteenä oman teeman tai lisäosan nimen kirjainlyhenne. Tämä palvelee kahta tarkoitusta: Ensinnäkin sillä vältetään mahdolliset yhteentörmäykset samannimisiä koukkuja tai funktioita käyttävien teemojen tai lisäosien kanssa. Toiseksi, kolmannen osapuolen kehittäjät, jotka haluavat käyttää koukkua omassa projektissaan, ymmärtävät paremmin koukun alkuperän ja tarkoituksen. [28.]

Suodatinkoukut

Toisin kuin toimintafunktioiden tapauksessa, suodatinkoukkuihin (engl. filter hook) kytkettyjen funktioiden tarkoitus on muuttaa tavalla tai toisella jo olemassa olevaa dataa. Rakennettaessa räätälöityä toiminnallisuutta WordPress-sivustolle on turvallista olettaa, että lähestulkoon kaikelle WordPressin prosessoimalle datalle on olemassa asianmukainen suodatinkoukku. On hyvin yleistä, että esimerkiksi artikkelien sisältödataa tulee jollain tietyllä tavalla muokata ennen sivulle tulostamista. Tähän tarkoitukseen on olemassa the_content-suodatinkoukku. Tähän koukkuun kytketyt funktiot suoritetaan siinä vaiheessa, kun artikkelin sisältödata on noudettu tietokannasta mutta sitä ei ole vielä tulostettu sivunäkymään. Esimerkkikoodissa 2 the_content-suodatinkoukkuun kytketty funktio muotoilee artikkelin sisältöä lisäämällä artikkelin pääkuvan tulostettavaksi sivulle ennen varsinaista sisältöä.

```
add_filter( 'the_content', 'featured_image_before_content' );

function featured_image_before_content( $content ) {
      if ( is_singular('post') && has_post_thumbnail()) {

            $thumbnail = get_the_post_thumbnail();

            $content = $thumbnail . $content;
      }

      return $content;
}
```

Esimerkkikoodi 2. the_content-koukku ja siihen kytketty suodatinfunktio [29].

Esimerkkikoodissa sisältöä muokataan ainoastaan siinä tapauksessa, jos käyttäjä on yksittäisen artikkelin sivulla ja tekijä on asettanut pääkuvan artikkelin editointinäkymässä. WordPress sisältää toistakymmentä ehtolausetta, joita yhdistelemällä teeman kehittäjä voi vaivattomasti ja tarkasti hallita, missä tilanteissa hänen mukautettua logiikkaansa

suoritetaan. On erityisesti huomioitava myös, että toimintafunktioista poiketen suodatinfunktion tulee aina palauttaa suodatettava data. Mikäli näin ei tehdä, luonnollisesti sisältöä ei tulostettaisi sivulle ja muut kyseistä suodatinkoukkua käyttävät lisäosat päätyisivät virhetilanteisiin. [29.]

Omatekoisen suodatinkoukun käyttö toimii periaatteessa samalla tavalla kuin toimintakoukun tapauksessa, mutta itse suodatinfunktioiden kutsuun oikeassa kontekstissaan käytetään apply_filters-funktiota seuraavalla tavalla:

```
$muuttuja = apply_filters( string $koukun_nimi, mixed $suodatettava, mixed
$argumentti1, ... );
```

Koska suodatinfunktio palauttaa aina suodatettavan datan, viittaus apply_filters-funktion arvoon tulee siis varastoida muuttujaan. Suodatettavan muuttujan ja vaihtoehtoisten suodatinparametrien tyypit määräytyvät käyttötilanteen mukaan. On syytä myös huomioida, että funktioon syötettyjen vaihtoehtoisten parametrien lukumäärän tulee vastata sitä kokonaislukuarvoa, joka määritellään vastaavan add_filter-funktion neljänneksi parametriksi. [30.]

Joissakin tilanteissa voi osoittautua tarpeelliseksi poistaa tietty toiminta- tai suodatinfunktio koukustaan. Tämä tehdään käyttämällä toiminnan tapauksessa funktiota remove_action ja suodattimella funktiota remove_filter seuraavasti:

```
remove_{action | filter}( string $koukun_nimi, callable $poistettava_funktio,
int $prioriteetti);
```

Poistofunktiota tulee kutsua funktion sisällä eikä suoraan teeman tai lisäosan tiedoston globaalilla näkyvyysalueella. Funktion toisen parametrin muoto vaihtelee myös sen mukaan, lisättiinkö poistettava funktio alun perin PHP-luokan sisällä vai ei. Mikäli koukkuunkytkentä tehtiin luokan sisällä, funktion poistoon tarvitaan muuttuja, jolla viitataan kyseisen luokan olioon. Kuvitellaan tilanne, jossa tietty lisäosa on lisännyt WordPress-ympäristöön olion, jonka sisällä lisätään artikkelin sisältöä jollain tapaa suodattava funktio. Poistofunktio kirjoitettaisiin silloin esimerkkikoodin 3 mukaisesti.

```
global $my_class;
remove_filter( 'the_content', array($my_class, 'class_filter_function') );
```

Esimerkkikoodi 3. Luokan kautta lisätyn suodatinfunktion poisto [31].

Mikäli luokka on staattinen, muuttujan sijasta käytettäisiin taulukon sisällä itse luokan nimeä. Olipa kyseessä toiminta- tai suodatinfunktio, jos funktion lisäyksessä käytettiin eksplisiittistä prioriteettiarvoa, myös poistofunktiossa on käytettävä tismalleen samaa prioriteetin lukuarvoa. Poistofunktio tulee mahdollisesti myös kiinnittää sellaiseen koukkuun, joka tapahtuu toiminta- tai suodatinfunktion lisäyksen jälkeen. Poistofunktiosta ei nimittäin olisi paljon juurikaan hyötyä silloin, kun itse poistettavaa toiminta- tai suodatinfunktiota ei ole vielä lisätty ympäristöön. [31; 32.]

3.4 Tietokantahallinta

MySQL on keskeinen osa LAMP-sovelluspinoa ja siten myös tärkeä kokonaisuus monissa PHP-ohjelmointikielellä toteutetuissa sisällönhallintajärjestelmissä ja verkkosovelluskehyksissä. Samalla tavalla kuin esimerkiksi Joomlan tai Drupalin tapauksessa, tämä avoimeen lähdekoodiin perustuva tietokantaohjelmisto toimii WordPress-sivuston ja sen osana toimivien teemojen ja lisäosien datan keskiössä. Tietokantakeskeisyys ajaa suositun verkkosovelluksen kehityksen väistämättä suuntaan, jossa kehittäjillä on käytössään yhä laajempi joukko erilaisia tietokantadatan käsittelyä helpottavia työkaluja ja rajapintoja. Pienen kokoluokan WordPress-sivuston kehityksessä on varsin epätodennäköistä, että kehittäjälle tulisi tarve sen syvemmin perehtyä SQL-ohjelmointiin suorittaakseen tavallisia tietokantadataa käsitteleviä operaatioita.

Tavanomaisen WordPress-asennuksen tietokanta koostuu 11 taulusta, jotka keskenään muodostavat varsin selkeän ja suoraviivaisen kokonaisuuden. WordPressin tietokannassa useat taulut on linkitetty toisiinsa one-to-many-relaatioilla, millä on pyritty vähentämään tallennettavan datan kokonaismäärää. Tämä tarkoittaa käytännössä sitä, että jos henkilö on kirjoittanut ja tallentanut tietokantaan useita artikkeleita, hänen käyttäjädataansa ei tallenneta toistuvasti yhä uudelleen tietokantaan jokaisen tallennettavan artikkelin kohdalla. Sen sijaan artikkelin datan varastoiva taulu sisältää vierasavainkentän, johon tallennetaan kirjoittajan käyttäjädataan viittaava tunnusluku. Väistämättä tärkein taulu tietokannassa on wp_posts. Koska WordPress-teemoissa ja -lisäosissa on mahdollista määritellä artikkelin käsittävän periaatteessa melkein minkätyyppistä sisältöä tahansa, wp_posts-taulu voi tilanteesta riippuen sisältää kirjavan määrän erityyppistä dataa. Kuitenkin se, mitä yleensä aina vähintään varastoidaan wp_posts-tauluun, ovat tavalliset artikkelit, sivut, liitemedia, navigaatiomenun oliot sekä kaikkien sivustolle rekisteröityjen mukautettujen artikkelityyppien data. [33.]

Lisäksi on tärkeää ymmärtää, mitä tarkalleen ottaen on artikkelin metadata ja mikä on metadatan rooli teema- ja lisäosakehityksen näkökulmasta. WordPress-artikkelien metadatasta puhuttaessa yleensä tarkoitetaan artikkelin sisällön mukana tallennettavia, mielivaltaista informaatiota sisältäviä mukautettuja kenttiä, jotka tallennetaan wp_postmeta-tauluun. Yksi wp_postmeta-taulun olio sisältää kentät artikkelin ja itse metadatan tunnusluvuille ja metadatan avain-arvoparille. Artikkeliin tallennetun metadatakentän arvon kyselyyn ja muokkaukseen voidaan käyttää funktioita [add | get | update | delete]_post_meta. Vain mielikuvitus lienee rajoitteena sille, millaisiin käyttöskenaarioihin ja ongelmiin artikkelin metadatakentät ovat sovellettavissa. Itse metadatan arvon tulostamisen lisäksi dataa voidaan luonnollisesti hyödyntää artikkelin sisällön esittämiseen ja suodattamiseen liittyvän logiikan rakentamisessa.

Ammattimaisissa verkkokehitysprojekteissa saattaa toisinaan tulla vastaan tilanteita, joissa useat WordPressiin sisäänrakennetut tietokantakyselyitä abstrahoivat työkalut, kuten wp_insert_post-funktio tai WP_Query-luokka, eivät ole enää riittäviä ja tarvitaan suorempaa tapaa keskustella tietokannan kanssa. Ongelma voi ilmetä esimerkiksi tilanteessa, jossa kehitettävän lisäosan data ei monimutkaisen rakenteensa puitteissa sovellu toteutettavaksi tavanomaisena artikkelityyppinä, vaan edellyttää tallentamista erilliseen, dedikoituun tietokantataulun. wpdb on WordPressin tärkein tietokantaoperaatioita suorittava yksittäinen luokka. wpdb-luokan pääasiallisin tarkoitus on tarjota rajapinta WordPress-tietokantadatan käsittelyä ja SQL-kyselyiden suorittamista varten, vaikkakin tämä luokka voidaan konfiguroida keskustelemaan myös minkä tahansa muun ulkoisen MySQL-tietokannan kanssa. [34; 35.]

wpdb-luokan metodeita kuuluu kutsua ainoastaan WordPressin alustaman globaalin muuttujan, $wpdb, kautta. Tämä muuttuja sisältää viittauksen luokan wpdb instanssiin. Jotta $wpdb-muuttujaa voitaisiin käyttää koodissa, se tulee määrittää viittaamaan oikeaan globaaliin muuttujaan avainsanalla global tai käyttämällä PHP:n superglobaalia $GLOBALS. wpdb-luokkaa voidaan käyttää lukemaan ja muokkaamaan dataa käytännössä minkä tahansa mielivaltaisen tietokantataulun tapauksessa. Oletuksena wpdb-luokan ei kuitenkaan ole mahdollista keskustella yhtä aikaa useamman kuin yhden tietokannan kanssa. Mikäli syystä tai toisesta tämä katsotaan tarpeelliseksi, wpdp-luokasta voidaan myös luoda erillinen instanssi, jolle on syötetty toisen tietokannan tunnukset ja yhteystiedot. Yhteyden muodostaminen ulkoiseen tietokantaan käyttämällä wpdb-luokan

instanssia on suositeltavaa siitä näkökulmasta, että kaikki wpdp-luokan tietokantaoperaatioita suorittavat metodit ovat vapaasti käytettävissä. Ulkoisen tietokantayhteyden ja ja kyselyn suorittaminen tapahtuisi esimerkkikoodin 4 tavoin.

```
$mydb = new wpdb( 'käyttäjätunnus', 'salasana', 'tietokannan_nimi', 'osoite'
);

$mydb->query('DELETE FROM ulkoinen_taulu WHERE id = 1');
```

Esimerkkikoodi 4. Yhteyden muodostaminen ulkoiseen tietokantaan konfiguroimalla uusi wpdb-luokan olio [34].

Vaikka wpdb-luokan metodilla query voidaan suorittaa käytännössä mitä tahansa mielivaltaisia SQL-kyselyitä, on suositeltavaa hyödyntää wpdb-luokkaan määriteltyjä kenttiä ja apumetodeita tietokantakyselyiden formuloimiseen. Tämä pätee erityisesti niissä tapauksissa, joissa käsitellään ainoastaan WordPress-tietokannan oletustauluja. Merkittävänä etuna wpdb-luokan tietokantaoperaatioita suorittavien apumetodien käytössä on se, että erilaisia datan validointi- ja sanitointitoimenpiteitä suoritetaan automaattisesti taustalla [34]. Esimerkkikoodi 5 esittää yksinkertaisen wpdb-olion käyttötapauksen, jossa tallennetaan artikkelin metadataa wp_postmeta-tauluun.

```
$global wpdb;

$post_id    = $_POST['post_id'];
$meta_key   = $_POST['meta_key'];
$meta_value = $_POST['meta_value'];

$wpdb->insert(
  $wpdb->postmeta,
  array(
    'post_id'    => $_POST['post_id'],
    'meta_key'   => $_POST['meta_key'],
    'meta_value' => $_POST['meta_value']
  )
);
```

Esimerkkikoodi 5. wpdb-luokan insert-metodin käyttö tietokantataulun rivin tallennukseen [34].

Esimerkkikoodissa insert-metodissa viitataan wp_postmeta-tauluun käyttämällä wpdb-luokan kenttää postmeta. Tämän menettelyn etuna on, että taulujen nimet tai etuliitteet eivät ole kovakoodattuna tietokantakyselyihin. Huomioitavaa edeltävässä esimerkissä on myös se, että syötetty data sanitoidaan oletuksena merkkijonona, koska insert-metodin kolmanneksi parametriksi ei ole määritetty eksplisiittistä sanitointiformaattia. Käytettäessä wpdb-luokan metodeita query ja get_results on tärkeää varmistaa, että sovelluskoodi on suojattu mahdollisilta SQL-injektiohyökkäyksiltä. Ongelma voidaan välttää käyt-

tämällä edellä mainittujen metodien yhteydessä kolmatta wpdb-metodia, prepare. Esimerkkikoodi 6 esittää prepare-metodin käytön tilanteessa, jossa artikkelin metadatatietue tulee poistaa wp_postmeta-taulusta.

```
$global wpdb;

$post_id = $_POST['post_id'];
$key     = $_POST['meta_key'];

$wpdb->query(
  $wpdb->prepare(
    "DELETE FROM $wpdb->postmeta
    WHERE post_id = %d
    AND meta_key = %s",
    $post_id,
    $key
  )
);
```

Esimerkkikoodi 6. wpdb-luokan prepare-metodi ajetaan ennen varsinaista query-metodin suorittamaa tietokantakyselyä [34].

Prepare-metodin ensimmäinen parametri sisältää itse suoritettavan SQL-kyselyn, jossa kenttien arvoja edustavat symbolit korvataan sanitoiduilla muuttujien $post_id ja $key arvoilla. Metodi tukee kolmea eri sanitointiformaattia: %s merkkijonoille, %d kokonaisluvuille ja %f liukuluvuille.

4 Verkkokehittämisen hyvät käytänteet

4.1 Yleiset menettelytavat

4.1.1 Kehitysympäristö ja versionhallinta

Verkkoviestinnän varhaisina aikoina oli tavallinen käytäntö muokata sivuston koodia suoraan tuotantoympäristöön. Tämä tehtiin muodostamalla FTP-yhteys verkkopalvelimelle, jolloin koodipäivitykset kirjoitettiin tekstieditorilla sivuston tiedostoihin. Tällä menettelyllä saatetaan toimia toisinaan jopa tänäkin päivänä, erityisesti pienten sivustojen ja harrastelijakoodareiden projektien tapauksessa. Menettely on toki helppoa ja nopeaa, eikä pienempien sivustojen ja yksinkertaisten muutosten kohdalla yleensä ole perusteltua tehdä laajoja esivalmisteluja. Sen sijaan, kun kysymys on esimerkiksi verkkokauppasivustosta, jonka tulee toimia luotettavasti ja ilman käyttökatkoja, sovelluspäivitysten tulee olla testattavissa ilman, että sivuston tavanomainen käyttö häiriintyisi.

Verkkosivuston tai -sovelluksen kehityksen elinkaarta voidaan käsitellä korkealla tasolla kahden eri ympäristön näkökulmasta: **kehitysympäristö** (engl. development environment, puhekielessä dev) ja **tuotantoympäristö** (engl. production environment, puhekielessä prod). Kehitysympäristö jaotellaan edelleen kehittäjän omalle tietokoneelle, eli localhostiin, asennettuun paikalliseen palvelinohjelmistoon ja toiselle palvelinkoneelle tai pilvipalveluun asennettuun testiympäristöön (engl. staging environment). Useimmissa tapauksissa tuotantoympäristö taas vastaa käytännössä Internet- tai webhotellipalveluntarjoajalta ostettua verkkopalvelintilaa, vaikkakin kyseessä voi olla myös yrityksen itse ylläpitämä, dedikoitu verkkopalvelin.

Koska missä tahansa ammattimaisessa verkkokehittämisessä on oltava mahdollisuus testata sivustoon tehdyt muutokset ennen päivitystä tuotantopalvelimelle, vähimmäisvaatimuksena on valmistella paikallinen palvelinympäristö kehittämistä ja testausta varten. PHP-sovellusten tapauksessa varsin yleinen tapa on asentaa tietokoneelle käyttöjärjestelmästä riippuen LAMP-ohjelmistopinoon pohjautuva sovelluskokoelma. Windowsin tapauksessa voidaan käyttää WampServer-ohjelmistoa, macOS:llä vastaava olisi MAMP. Käyttöjärjestelmästä riippumaton vaihtoehto olisi XAMPP. Kaikkien kolmen tapauksessa periaatteena on, että ohjelmisto asennetaan kehittäjän tietokoneelle ja se toimii lähestulkoon samalla tavalla kuin mikä tahansa muukin WWW-palvelin. Ohjelmistoon on asennettu valmiiksi Apache HTTP -palvelinohjelmisto, MySQL-tietokantaohjelmisto ja PHP. Tällaisen sovelluskokoelman asentaminen ja käyttöönotto on yleensä verrattain nopeaa ja sopii pienen budjetin kehitysprojekteihin. Toisaalta kehittäjän vastuulle jää huolehtia, että kokoonpano sisältää juuri ne samat komponentit, ohjelmistomoduulit ja konfiguraatiot, jotka ovat asennettuna tai määritettynä sivuston tuotantopalvelimella.

Paikallinen kehitysympäristö voidaan vaihtoehtoisesti virtualisoida siten, että testattava sovellus toimii täysin irrallisena ja riippumattomana itse työympäristön järjestelmästä ja siihen asennetuista ohjelmista. Merkittävänä ongelmana tässäkin lähestymistavassa on kuitenkin se, että tuotantoympäristöä vastaavan palvelinsovellusten ja -moduulien kokoaminen siten, että sovelluspino vastaa mahdollisimman identtisesti lopullista tuotantoympäristöä, on aikaa vievää prosessi. Virtuaalisen kehitysympäristön valmisteluun on kuitenkin olemassa työkaluja, joilla virtuaalipalvelinsovellusten ja -moduulien asentaminen ja konfigurointi voidaan automatisoida käyttämällä ennalta määriteltyjä ohjeita tai malleja. Esimerkkinä tällaisista työkaluista mainittakoon Vagrant, jolla automatisoidaan monia virtuaalisen kehitysympäristön konfiguraatioon liittyviä toimenpiteitä.

Rakenteeltaan Vagrant toimii käytännössä välikätenä palvelinvarusteluteknologian (engl. server provisioning) ja virtualisointiohjelman päällä. Tässä kokonaisuudessa palvelinvarustelusovellus, kuten Puppet tai Chef, rakentaa palvelinkokoonpanon virtualisointiohjelman, kuten VMware ESXi:n tai VirtualBoxin, sisälle alustettuun virtuaalikoneeseen. Vagrant siis nojautuu voimakkaasti palvelinvarustelu- ja virtualisointiteknologioihin, ja sen tehtävänä on edelleen yksinkertaistaa erilaisten palvelinkokoonpanojen asennus ja pakata nämä kokoonpanot uudelleenkäytettäviksi ja käyttöjärjestelmäriippumattomiksi paketeiksi. Käyttämällä näistä ohjelmista muodostuvaa kokonaisuutta voidaan verkkokehitysprojekteihin luoda itsenäisiä testiympäristöjä, joiden muokkaaminen ei vaikuta paikalliseen työympäristöön asennettuihin sovelluksiin ja joita voidaan helposti kopioida laitteelta toiselle. [36.]

Nykyään palvelinvirtualisointi ilmenee yhä näkyvämmin verkossa tarjotuissa **pilvilaskentapalveluissa**. Amazon.comin skaalautuvaa laskentakapasiteettia ja palvelininfrastruktuureja tarjoava tytäryhtiö, AWS (Amazon Web Services), on saavuttanut hyvinkin vahvan jalansijan Pohjoismaiden IT- ja verkkoviestintäpalveluiden tuotannossa. Useat suuret ja kasvavat suomalaiset yritykset, kuten Sanoma, Alma Media, Supercell ja DNA, hyödyntävät AWS-pilvipalvelinympäristön virtuaalikoneita ohjelmisto- ja verkkokehitysprojekteissaan [37].

Tämän raportin ja opinnäytetyön projektin näkökulmasta huomion arvoinen AWS-palvelu on niin kutsuttu Amazon Elastic Compute Cloud, lyhyemmin Amazon EC2. Tämä alusta tarjoaa kehittäjille nopean tavan määritellä ja käynnistää räätälöidyn palvelininstanssin, joka sisältää tarpeisiin sopivat laitteistoresurssit ja esiasennetut ohjelmistot. Se, millaiset laitteistoresurssit instanssi tarkalleen sisältää, riippuu valitun instanssin tyypistä. Amazon EC2-palvelun valikoimaan sisältyy joukko erilaisista laiteresursseista koostettuja instanssityyppejä, jotka on lajiteltu käyttötarkoituksen mukaan. Kukin instanssityyppi on edelleen jaettu kasvaviin suuruusluokkiin sen mukaan, kuinka paljon kuormankantokykyä instanssille halutaan. Mikäli instanssi on tarkoitus ottaa vain puhtaasti sovelluskehitys- ja testauskäyttöön eikä palvelimelle ole odotettavissa suurta kävijämäärää, tähän tarpeeseen mitä todennäköisimmin riittänee yksi yleiskäyttöinen T2-tyypin instanssi. Tämä tyyppi on hinta-laatusuhteeltaan edullinen matalan liikenteen sivustoille, jotka eivät kovinkaan usein käytä suorittimen maksimikapasiteettia mutta joiden toisinaan tulee lyhytaikaisesti syöksyä kapasiteettirajojen yläpuolelle. [38; 39.]

Kun kysymyksessä on ammatillinen, usean kehittäjän yhteistyönä tehtävä verkkokehitysprojekti, kehitysympäristön lisäksi luonnollisesti tarvitaan työkaluja, jotka helpottavat saman sovelluksen koodikantaan tehtävien muutosten hallintaa ja niiden systemaattista vientiä tuotantoympäristöön. Versionhallintaohjelmiston käyttö, tai vähintäänkin perusteiden ymmärtäminen, onkin modernin verkkokehittäjän tärkeä taito. Lyhyesti ilmaistuna, versionhallinta on järjestelmä, joka tallentaa kaikki sovelluksen kooditiedostoihin tehdyt muutokset, tarjoaa työkaluja eri ajankohtina tallennettujen tiedostojen seurantaan ja vertailuun ja mahdollistaa virheiden tai ongelmien ilmaantuessa sovelluksen palauttamisen aikaisempaan versioon. Versionhallintaohjelmistojen olemassaolo mahdollistaa sen, että kehittäjän ei tarvitse sovellusta tai sen komponenttia työstäessään muistaa tai ylläpitää erillistä listaa siitä, mihin tiedostoihin ja koodinpätkiin hän teki muutoksia minäkin annettuna ajankohtana.

Erilaisia versionhallinnan tekniikoita ja työskentelymalleja ilmenee monessa eri muodossa. Kaksi tunnetuinta versionhallintajärjestelmien arkkitehtuuria ovat keskitetty ja hajautettu versionhallinta. **Keskitetyn versionhallinnan** (engl. centralized version control) lähtökohta oli helpottaa kehittäjien välistä yhteistyötä ohjelmointiprojekteissa, joissa kehittäjät työskentelevät etänä omilla tietokoneillaan, mahdollisesti vieläpä toisistaan poikkeavilla käyttöjärjestelmäkokoonpanoilla. Tässä järjestelmässä varsinainen versionhallinta keskittyy yhden palvelimen säilöön, jonne koodikantaan tehtävien muutosten historia tallennetaan. Asiakaskoneet voivat muodostaa yhteyksiä tähän palvelimeen käyttäen esimerkiksi HTTP- tai SSH-protokollaa, ladata sovelluksen versiohistorian viimeisimmät tiedostot paikalliseen työympäristöön ja tallentaa tiedostoihin työstettyjä pysyviä muutoksia (puhekielessä ”committaa ja työntää”) etäsäilön versionhallintaan.

Keskitetyn versionhallinnan vahvuutena pidetään ohjelmointiprojektien helpompaa hallittavuutta verrattuna siihen tilanteeseen, jossa jokaisella kehittäjällä olisi ainoastaan oma paikallinen kopio sovelluksen versiohistorian sisältävästä tietokannasta. Keskitetystä mallista voidaan kuitenkin samalla helposti päätellä myös sen haittapuoli: mikäli etäsäilön sisältävä palvelin päätyy käyttökatkoon eikä säilöstä ole saatavilla väliaikaista kopiota, luonnollisesti kehittäjät eivät katkon aikana voi tallentaa muutoksia versionhallintaan.

Vuodesta 2005 eteenpäin ohjelmistotuotannon alueella yleistyi erilaisia versionhallintajärjestelmiä, joita alettiin kutsua nimellä **hajautetut versionhallintajärjestelmät** (engl.

distributed version control systems). Keskitetystä versionhallinnasta poiketen hajautetussa versionhallinnassa asiakaskoneet eivät kopioi pelkästään viimeisimpiä tiedostoja etäsäilöstä paikalliselle levylle, vaan samalla säilön versiohistoria tallennetaan kokonaisuudessaan paikalliseen tietokantaan. Toisin sanoen, kun kehittäjä lataa hajautetun versionhallinnan etäsäilöön varastoidut tiedostot omalle työkoneelleen, sovelluksen kaikkiin tiedostoihin tehtyjen muutosten historia tallentuu etäsäilön kanssa identtiseen paikalliseen säilöön.

Hajautetun arkkitehtuurin etuna on, että pysyvien muutosten tallentaminen versionhallintaan ei edellytä välitöntä verkkoyhteyttä etäpalvelimelle. Muutokset, kuten yksittäisen sovelluskomponentin tai -toiminnallisuuden päivitys, voidaan työntää etäsäilöön myöhemmin esimerkiksi paikallisen testauksen jälkeen. Hajautettu versionhallinta tarjoaa myös joustavuutta kehitysprojektien työnjaon ja palvelinarkkitehtuurin suhteen, sillä jokainen projektiin osallistuva kehittäjä voi ylläpitää säilöä, joka on samanaikaisesti potentiaalinen työpanos koko projektille ja enemmän tai vähemmän keskitetty etäsäilö muiden kehittäjien työpanoksille.

Raportin kirjoittamishetkellä kaikkein suosituin versionhallintajärjestelmä on epäilemättä Git. Olettaen, että maailman versionhallintajärjestelmien markkinat koostuisivat ainoastaan seuraavasta viidestä teknologiasta: Git, SVN, Mercurial, Perforce Helix ja CVS, Google Trends -analytiikkapalvelusta kerätyn datan perusteella Gitiin liittyvät Google-haut kattoivat noin 70 % kaikkien viiden edellä mainitun versionhallintajärjestelmän hauista vuonna 2016. Saman vuoden aikana Stack Overflow -sivustolla esitettiin yli 20 000 Gitiin liittyvää kysymystä, mikä vastaa noin 87 %:a kaikkien viiden edellä mainitun teknologian kysymysten yhteismäärästä. [40.]

Gitistä tekee poikkeavan suhteessa useisiin muihin versionhallintajärjestelmiin se, että tieto muutoksista ei tallennu säilöön yksittäisten tiedostojen joukkona ja kuhunkin tähän joukkoon kuuluvaan tiedostoon tehtyjen muutosten listana. Sen sijaan muutoshistoria koostuu peräkkäisistä tilannekuvista, joista jokaiseen on tallennettu koko projektin tiedostojärjestelmä ja viittaus tähän tilannekuvaan. Muistin ja laskennan säästämiseksi jo aikaisemmin tallennettuja tiedostoja ei tallenneta uudestaan tilannekuvaan, vaan sen sijaan tallennetaan linkki kyseiseen tiedostoon aikaisemmassa tilannekuvassa. Koska Git on hajautettu versionhallintajärjestelmä, muutoksia tallennetaan pitkälti paikalliseen säilöön ja suurin osa Gitin operaatioista on paikallisia. Yhteys etäsäilöön tulee muodostaa

ainoastaan silloin, kun työstetyt muutokset tulee työntää sinne tai viimeisin kopio etäsäilöstä tulee vetää paikalliseen säilöön.

Paikallinen prosessointi tekee työskentelystä nopeaa, sillä jos halutaan esimerkiksi verrata tiedoston sisältöä kolme pysyvää muutosta aikaisempaan versioon, poikkeavuuslaskennassa käytettävien versioiden sisältötiedot haetaan suoraan paikallisesta tietokannasta. Väitetään olevan lähes mahdotonta tehdä versionhallinnassa oleviin tiedostoihin muutoksia ilman, että Git siitä tietäisi. Tämä mahdollistetaan SHA-1-tiivistefunktion tuottamalla 40-merkkisellä heksadesimaalisella tarkistussummalla, joka lasketaan kulloisenkin muokatun tiedoston sisällön tai kansion rakenteen perusteella. Tätä tarkistussummaa käytetään lopulta jokaisen pysyvän muutoksen tallentamiseen ja viittaamiseen versiohistoriassa. [41, s. 10.]

Git-projekti voidaan jakaa käsitteellisesti kolmeen alueeseen, joiden sisäistäminen on ensisijainen edellytys Git-versionhallinnan sujuvaan peruskäyttöön ja tietopohjan kartuttamiseen. Itse Gitin paikallinen oliotietokanta ja metatieto tallentuu **Git-hakemistoon**, projektin juuressa sijaitsevaan piilotettuun kansioon .git. Tämä kansio kopioidaan joka kerta paikalliseen tietokoneeseen, kun git-projekti kloonataan etäsäilöstä. **Työskentelyhakemisto** on yksittäinen tiedonhaku (engl. checkout) jostakin tietystä projektin tiedostojen ja kansiorakenteen tilannekuvan versiosta. Työskentelyhakemistoon kuuluvat tiedostot ovat vedetty ulos pakatusta tietokannasta, ja ne ovat valmiita muokattavaksi kehittäjän suosimalla editorilla. **Valmistelualue** on Git-hakemistossa sijaitseva tiedosto, johon tallennetaan informaatiota siitä, mitkä muutokset sisällytetään seuraavaan versiohistorian pysyvään muutokseen. Oletetaan esimerkkinä tilanne, jossa kehittäjän tulee työntää tehdyt koodimuutokset etäsäilössä istuvaan Git-projektiin. Olettaen, että Git on asennettuna paikalliseen tietokoneeseen, joka on yhteydessä Internetiin, versionhallinnan perustyönkulku voidaan jakaa seuraaviin vaiheisiin:

1. Etäsäilö kloonataan paikalliseksi kansioksi navigoimalla haluttuun polkuun komentotulkkiohjelmalla ja suorittamalla komento git clone [etäsäilön osoite]. Projektikansioon navigoidaan komentotulkkiohjelmalla, ja tiedostoja aletaan muokata Gitin työskentelyhakemistossa.

2. Tiedostoihin ja kansiorakenteeseen kohdistuneita muutoksia seurataan suorittamalla git status -komento. Halutut muutokset voidaan lisätä valmistelualueeseen

suorittamalla komento git add [tiedoston nimi]. Mikäli kaikki muutokset halutaan lisätä valmistelualueeseen, suoritetaan komento git add . (piste).

3. Valmistelualueeseen lisätyt muutokset tallennetaan pysyvästi paikalliseen versiohistoriaan suorittamalla komento git commit -m ”[muutosta koskeva kuvausviesti]”. Lopuksi muutokset työnnetään etäsäilöön suorittamalla komento git push.

Koska projekti kloonattiin vaiheessa 1, etäprojekti (remote) ja haara asetettiin implisiittisesti arvoihin origin ja master, eikä niitä siis tarvitse eksplisiittisesti määritellä kommennolla git push | git pull [etäprojektin nimi] [haaran nimi]. Mikäli toinen kehittäjä on ennättänyt työntää uusimmat muutoksensa etäsäilöön ennen vaihetta 3, Git hylkää työnnön etäsäilöön. Työntö onnistuu siinä vaiheessa, kun uusimmat muutokset on ensin vedetty paikalliseen säilöön ja kahden eri tahon työstämät muutokset yhdistetään (engl. merge) ilman konflikteja. [41, s. 10; 41, s. 43–48.]

4.1.2 Suorituskyvyn optimointi

Nopeasti toimiva ja latautuva verkkosovellus on ensiarvoisen tärkeä tavoite kenelle tahansa verkkoviestintää harjoittavalle. Sivuston suorituskyvyllä ja nopealla vasteajalla on välitön vaikutus sen käytettävyyteen ja tätä myötä edelleen tuotettuun verkkoliikenteeseen ja sivuistuntojen pituuteen. Tässä asiayhteydessä sivuston tai sovelluksen jouhevuus ja sulavuus ei käsitä yksistään sitä, kuinka nopeasti yksittäinen sivu latautuu, vaan siihen liittyy myös esimerkiksi käyttöliittymäinteraktioiden ja -animaatioiden viive sekä erilaisten HTML DOM -elementtien ja CSS-tyylimäärittelyjen renderöintinopeus selaimessa.

Paul Irishin ja Paul Lewisin [42] esittämä RAIL-käytettävyysmittari antaa mielikuvaa sen suhteen, millaiset viiveajat ovat verkkokäyttäjien näkökulmasta hyväksyttäviä. Erilaisten grafiikoiden ja efektien animaatiot mielletään sulaviksi, jos näytölle päivitetään 60 kehystä sekunnissa eli noin yksi uusi kuva joka 16 millisekunti. Kun käyttäjä klikkaa painiketta, käyttöliittymän tulisi tuottaa käyttäjälle vastaus tai päivittää näkymä alle 100 millisekunnissa. Uuden sivun latautumisen odotetaan tapahtuvan alle sekunnissa. Yli sekunnin mittaisessa viiveessä käyttäjä menettää keskittymisensä tiettyyn toimintoon, ja 5–10 sekuntia kestävässä odotuksessa käyttäjä turhautuu ja mitä todennäköisimmin luovuttaa eikä palaa enää yrittämään uudelleen. [42.]

Erilaisten sivuston verkkoaineistojen, kuten mediatiedostojen ja CSS- ja Javascript-tiedostojen lataamisen optimointi ja sivulatauksen aikana lähetettävien HTTP-pyyntöjen lukumäärän karsiminen minimiin ovat tärkeä lähtökohta suorituskyvyn optimoinnissa. Niiden tiedostojen, jotka on välttämätöntä ladata sivulle, tulee datan formaatista riippuen olla ajettuna jonkinlaisen pakkausprosessin läpi tai muuten esikäsiteltynä verkkoon sopivaksi. Useissa tapauksissa kuvatiedostot kattavat ylivoimaisesti suurimman osan kaikesta datasta, jonka käyttäjän selain lataa yhden sivulatauksen aikana. Luonnollisesti siis tarpeettoman kuvadatan välttäminen, kuvien pakkaaminen verkkokäyttöön sopivaksi ja kuvan oikean esitysmuodon tunnistaminen tiettyyn käyttötapaukseen ovat välttämättömiä huomionaiheita optimoinnin näkökulmasta.

Tavallisimmat kuvaformaatit verkkokäyttöön ovat bittikarttakuvista JPEG, PNG ja GIF ja vektorigrafiikkakuvista SVG ja fonttikuvakkeet. Mikäli kuvaa halutaan käyttää esimerkiksi käyttöliittymäkuvakkeena, jolloin kuvassa ei ole sallittua tapahtua pikselöitymistä riippumatta päätenäyttölaitteen esitystarkkuudesta, ei yleensä ole perustelua käyttää bittikarttagrafiikkaa. Bittikarttakuvat eivät pikselirakenteensa vuoksi skaalaudu hyvin eri näyttöresoluutioille. Olettaen, että korkean resoluution ja pikselitiheyden näyttölaitteet (esimerkkinä Applen markkinoimat retinanäytöt) yleistyvät verkon selaamisessa, bittikarttakuvien tapauksessa tarvitaan samoista kuvista yhä suuremman resoluution versioita, jotta kuvat näyttäisivät näyttölaitteilla laadukkailta. Tämä luonnollisesti tarkoittaa enemmän verkon kautta ladattavaa kuvadataa, vaikka kuvatiedostot pakattaisiinkin asianmukaisesti verkkokäyttöön.

Yksinkertaisista muodoista koostuvat graafiset elementit, kuten kuvakkeet, logot ja kaksiulotteiset kuvitukset, on tehokkaampaa toteuttaa joko puhtaalla CSS-koodilla tai käyttämällä vektorigrafiikkaa, kuten SVG-kuvia ja fonttikuvakkeita. SVG on XML-pohjainen kuvaformaatti, jossa kuvan muodostuminen perustuu, kuten muissakin vektorigrafiikoissa, pikselimatriisin sijaan bézier-käyristä muodostettuihin geometrisiin muotoihin. Olettaen, että verkkosivulla käytettävä graafinen elementti on kuvainformaation puitteissa yksinkertainen, on todennäköistä, että grafiikan toteuttaminen SVG-muodossa tulee olemaan tiedostokooltaan kevyempi verrattuna vastaavaan bittikarttakuvaan. Koska sama SVG-kuva skaalautuu vektorigrafiikkana mille tahansa resoluutiolle ilman, että kuvan laatu heikkenee, ladattavan kuvadatan näkökulmasta kuva on hyvinkin optimaalinen eikä siitä tarvitse tuottaa ja esittää usean kokoisia versioita. SVG-grafiikan käytön yleistymiseen käyttöliittymäsuunnittelussa vaikuttaa myös sen XML-pohjainen rakenne. Koska SVG noudattaa DOM-standardia, kuten tavanomainen HTML-dokumenttikin,

SVG-kuvien ulkomuotoa ja ominaisuuksia on helppo käsitellä käyttämällä CSS- ja JavaScript-koodia. [43; 44.]

Verkkoaineistojen oikean esitysmuodon ja pakkausmenetelmän tunnistamisen lisäksi erilaisten välimuistien hyödyntämisellä resursseihin on merkittävä vaikutus sivuston suorituskykyyn. Lähtökohtaisesti tulee olettaa, että verkkoresurssien pyytäminen ja lataaminen palvelimelta ovat kalliita ja aikaa vieviä prosesseja. Välimuistin tehtävä on varastoida verkkosovelluksen sivut, mediatiedostot ja aineistot sellaiseen staattiseen muotoon ja sijaintiin, joista ne on mahdollista nopeasti uudelleenkäyttää siirryttäessä sivulta toiselle tai palattaessa sivulle takaisin.

Välimuistisovelluksia on useantyyppisiä, ja ne voivat sijaita asiakaspäässä, isäntäpalvelimella tai jollakin erillisellä välityspalvelimella. Verkkovälimuistia hyödyntämällä voidaan verkkopalvelimen HTTP-vastausten otsakkeiden avulla määritellä erilaisille lähetettäville resursseille käytänteet, joiden mukaan resurssi tallennetaan erilaisiin välimuisteihin. Verkkovälimuistin hyödyntäminen on siinä mielessä joustavaa, että resurssi voidaan varastoida uudelleenkäyttöä varten käyttäjän selaimen välimuistin lisäksi yhdelle tai useammalle välityspalvelimelle, jotka sijaitsevat lähempänä käyttäjää kuin lopullinen verkkopalvelin.

Seuraavassa on laajasti tuetun Cache-Control -otsakkeen erilaisia asetusparametreja, jotka voidaan konfiguroida palvelinasetuksiin, esimerkiksi Apachen httpd.config-tiedostoon:

- No-cache: välimuistiin tallennettu resurssi tulee uudelleenvalidoida jokaisen HTTP-pyynnön kohdalla, ennen kuin resurssi voidaan lähettää asiakkaalle. Vaikka resurssi merkitäänkin aina vanhentuneeksi, validointimekanismin hyödyntämisellä vältetään resurssin toistuva lataus palvelimelta.
- No-store: lähetettävää resurssia ei tule missään tapauksissa tallentaa välimuisteihin. Parametria tulee käyttää sellaisten resurssien kohdalla, jotka sisältävät arkaluontoista dataa.
- Public / private: mikäli resurssi on merkitty julkiseksi, se voidaan vapaasti tallentaa asiakkaan ja verkkopalvelimen välissä sijaitsevien välityspalvelimien välimuisteihin. Jos resurssi merkitään yksityiseksi, sallitaan sen tallentaminen ainoastaan asiakkaan verkkoselaimen välimuistiin.
- Max-age: määrittää resurssille enimmäisajan välimuistiin varastoimista varten. Kun resurssin ikä ylittää enimmäisajan, resurssi tulee joko validoida tai ladata uudelleen verkkopalvelimelta. Aika määritellään sekunteina, ja suurin sallittu arvo on 31 536 000 (1 vuosi).
- Must-revalidate: parametri edellyttää, että resurssin tuoreutta koskettavaa informaatiota, joka on määritelty esimerkiksi max-age-parametrin arvossa,

> tulee noudattaa tarkasti. Vanhentunutta sisältöä ei tule palauttaa välimuistista missään tapauksessa. Tällä voidaan estää välimuistiin tallennetun resurssin käyttö esimerkiksi erilaisten verkkohäiriöiden tapauksessa. [45.]

WordPress-sivustojen tapauksessa erilaisten välimuistilisäosien, kuten W3 Total Cache tai WP Super Cache, käytöstä on muodostunut jo tapa, josta ei ole syytä poiketa. Monilla WordPress-sivustoilla välimuistilisäosan toiminta onkin merkittävä, ellei suurin, suorituskyvyn optimointiin vaikuttava yksittäinen tekijä. Useat välimuistilisäosat hyödyntävät WordPress-ytimen välimuistijärjestelmää, johon sisältyvät muun muassa oliovälimuisti ja transienttirajapinta. Tämän lisäksi välimuistilisäosa saattaa hyödyntää useita palvelinpään optimointitekniikoita, kuten dynaamisesti koostettujen sivujen tallentamista staattisiksi kopioiksi palvelimen kiintolevylle tai keskusmuistiin, tietokantaolioiden tallentamista palvelimen kiintolevylle, opcode- tai memcached -muisteihin sekä skriptitiedostojen automaattista tiivistämistä ja pakkaamista. [46; 47.]

Optimoinnin kannalta on tärkeää huomioida, että nopeuden parantaminen ei koske pelkästään verkkoaineiston latausnopeuden ja verkkopyyntöjen lukumäärän vähentämistä minimiin; olennaista on myös pohtia, miten erilaisia raskaita tietokantapyyntöjä, ulkoisia rajapintakyselyitä ja palvelimen suorittamia vaativia laskentoja voidaan varastoida uudelleenkäytettävään muotoon. Koska WordPress on hyvin tietokantakeskeinen sovellus, suuren liikenteen sivustojen kohdalla on välttämätöntä varmistaa, etteivät palvelinresurssit ylikuormitu suosittuun artikkeliin tai muuhun dataan kohdistuneista tietokantapyynnöistä. Tietokantapyynnöt ja palvelinpään laskennat on siis aiheellista nähdä optimoitavina resursseina, siinä missä asiakkaalle lähetettävät kuva- tai skriptitiedostotkin.

Oliovälimuisti on ollut sisäänrakennettuna WordPressiin jo vuodesta 2005 lähtien. Ajatuksena on, että kyselyitä tietokantaan vähennettäisiin tallentamalla mielivaltainen data palvelimella suoritettavan PHP-sovelluksen muistiin. Data tallennetaan muistiin avain-arvo-parina siten, että datan sisältö on nopeasti noudettavissa muistista avaimen avulla. Huomioitavaa WordPressin oliovälimuistissa on kuitenkin se, että oletuksena tallennetun datan elinikä on vain yksi sivupyyntö. Mikäli varastoidun datan halutaan olevan käytettävissä yli yhden tai useamman sivupyynnön, toinen vaihtoehto olisi käyttää WordPressin transientteja. [47; 48.]

Transientit sisältävät mielivaltaista dataa, joka tallennetaan tietokantaan wp_options-tauluun samalla tavalla kuin WordPress-sivuston asetukset. Transientit poikkeavat asetuksista siinä, että ne sisältävät ylimääräisen kentän datan voimassaoloajan määrittämistä

varten. WordPress poistaa vanhentuneita transientteja tietokannasta epäsäännöllisin väliajoin. Mielenkiintoinen yksityiskohta transienttien tallennuksessa on, että voimassaoloaika sisältää ainoastaan maksimin muttei minimiä. On täysin mahdollista, että transienttidata on poistunut tietokannasta ennen voimassaoloajan päättymistä tai että sitä ei ole tallennettu kantaan alun perinkään. [49.]

Lähtökohtana välimuistilisäosien toiminnassa on, että WordPressin oliovälimuistia tai transientteja voidaan hyödyntää siten, että uudelleenkäytettävä dataa voidaan tallentaa väliaikaisesti, mutta niin pitkäksi aikaa kuin halutaan ja tehokkaammin verrattuna tietokantaan tallentamiseen. Käyttämällä esimerkiksi Memcached-ohjelmistoa, lisäosan on mahdollista tallentaa tietokantadataa, rajapintakyselyitä tai sivurenderöintejä lyhytaikaisesti palvelimen keskusmuistiin, josta data on noudettavissa uudelleen hyvin nopeasti. [50; 51.]

4.1.3 Sivuston responsiivisuus

Verkkoselaamisen siirtyminen mobiiliin ja erilaisten mobiililaitteiden valtava lukumäärä haastaa verkkokehittäjiä ja -suunnittelijoita tuottamaan verkkopalveluita ja käyttökokemuksia, jotka takaavat sivuston käytettävyyden ja esteettömyyden mahdollisimman monille erilaisille ja erikokoisille päätelaitteille, verkkoselaimille ja erilaisille verkon selaamisen olosuhteille. Responsiivisen suunnittelumallin periaatteena on, että sivuston sisältöelementtien muodot mukautuvat nesteenomaisesti käyttäjän selainikkunan leveyden (joissakin tapauksissa myös korkeuden) mukaan. Adaptiivisuudella taas viitataan siihen, että sisältöelementtien muoto, ulkoasu ja toiminnallisuus muuttuvat ainoastaan siirryttäessä tietynkokoiselta selainikkunan resoluutioalueelta toiselle. Täysin responsiivisen tai adaptiivisen sivuston toteuttaminen on vaikeaa ja epäkäytännöllistä. Mobiiliystävällisten sivustojen ulkoasun ja rakenteen suunnittelussa hyödynnetäänkin yleensä molempia lähestymistapoja saumattomasti.

Ethan Marcotten mukaan responsiivisten sovellusten kolme keskeistä komponenttia ovat **fluid grid**, **skaalautuvat kuvaelementit** ja **CSS-mediakyselyt**. Responsiivisten sivuelementtien toteuttamiseen riittää useimmiten HTML5- ja CSS3-tekniikoiden hyödyntäminen. Responsiivisuuden olennainen hyöty on sen kustannustehokkuus, sillä yrityksen ei tarvitse investoida verkkosivustojen erillisten mobiiliversioiden kehittämiseen ja ylläpitoon. Responsiivisuuden lähtökohtana on laiteagnostisuus: toiminnallisuutta ja sisällön

rakennetta pyritään muuttamaan progressiivisesti siirryttäessä erilaisiin selauskonteksteihin. Responsiivisten elementtien dimensiot määritellään käyttämällä suhteellisia CSS-yksiköitä, kuten %, VW ja VH. [52.]

CSS:n kolmannen version myötä mediakyselyt ovat tarjonneet frontend-kehittäjille varsin mutkattoman tavan kohdentaa tyylisääntöjä perustuen erilaisiin päätelaitteen ominaisuuksiin. Tällaisiin ominaisuuksiin kuuluvat muun muassa näytön leveys ja korkeus, orientaatio, kuvasuhde, resoluutio, näyttölaitteen värien määrä sekä päätelaitteen mediatyyppi. Verkkokehittämisen ja responsiivisuuden näkökulmasta käytännössä ainoa huomionarvoinen mediatyyppi on screen, sillä modernit älypuhelimet ja tabletit eivät lainkaan kuuntele mediatyyppiä handheld [53]. Responsiivisessa suunnittelussa keskitytäänkin pääasiallisesti sivuston selainikkunan dimensioihin.

Kehitystyön selkeyttämiseksi on aiheellista saavuttaa yksimielisyys siitä, mikä on responsiivisten tyylimäärittelyjen yleinen kulkusuunta. Toisin sanoen, missä järjestyksessä eri mediakyselyiden raja-arvot priorisoidaan tyylitiedostoissa. Mikäli projektissa sovelletaan niin kutsuttua mobile first -lähestymistapaa, seuraavat ehdot ovat voimassa:

- Pienimpiä mobiililaitteita koskettavat tyylisäännöt kirjoitetaan tyylitiedoston alkuun käyttämättä erillistä mediakyselyä. Tiedoston alkuun kirjoitetaan myös yleiset, kaikkia laitteita koskevat tyylisäännöt.
- Mediakyselyissä esiintyvät leveyssuuntaiset raja-arvot ilmoitetaan käyttämällä min-width-määritettä.
- Leveyssuuntaiset raja-arvot esiintyvät tyylitiedostossa suuruusjärjestyksessä pienimmästä suurimpaan.

Esimerkkikoodi 7 havainnollistaa, kuinka tunnetun verkkokehyksen, Bootstrapin, version 3.X less-tiedostoissa määritetään mediakyselyillä raja-alueet responsiivisen grid-näkymän tyylisäännöille.

```
/* Pienimpien mobiililaitteiden näytöt */
/* (älypuhelimet, näytön leveys pienempi kuin 768px) */
/* Tähän tulee oletustyylit, ei käytetä mediakyselyä */

/* Pienet näytöt (tabletit, 768px ja suuremmat) */
@media (min-width: @screen-sm-min) { ... }

/* Keskikokoiset näytöt (pöytätietokoneet, 992px ja suuremmat) */
@media (min-width: @screen-md-min) { ... }

/* Suuret näytöt (suuret pöytätietokoneet, 1200px ja suuremmat) */
@media (min-width: @screen-lg-min) { ... }
```

Esimerkkikoodi 7. Bootstrap 3.X -kehyksen grid-näkymän ja käyttöliittymäkomponenttien tyylittelyssä käytetyt mediakyselyt [54].

Mobile-first-periaatteen soveltamisesta argumentoidaan olevan hyötyä niin sivuston suorituskyvyn kuin CSS-koodin luettavuuden ja ylläpidon kannalta. On varsin tavallista törmätä tilanteisiin, joissa aiemmin määritettyjä tyylisääntöjä yliajetaan myöhemmin pienempien näyttöjen kontekstissa. Oletetaan tilanne, jossa pöytäkonenäkymän sisältöalue vie 60 % selainikkunan tilasta ja tämä alue on asetettu sivupalkin rinnalle vasemmalle puolelle säännöllä float: left. Mobiilinäkymässä sisältöalue ja sivupalkki taas ovat koko selainikkunan pituisia. Esimerkkikoodissa 8 on esitetty kaksi sass-koodinpätkää, ensimmäisessä esimerkissä mobiilityylit on eristetty oletustyyleistä mediakyselylohkolla, jossa raja-arvo ilmaistaan määritteellä max-width. Toisessa esimerkissä taas järjestys on päinvastainen ja pöytäkonetyylit on erotettu omaan mediakyselylohkoon min-width-määritteellä.

```
.content {
  // Tyylimäärittelyt pöytätietokonenäytöille.
  float: left;
  width: 60%;

  // Tyylimäärittelyt mobiilinäytöille.
  // Elementille tulee määrittää oletusominaisuudet erikseen.
  @media (max-width: 768px) {
    float: none;
    width: 100%;
  }
}

.content {
  // Tyylimäärittelyt mobiilinäytöille.
  // Tänne ei tarvitse kirjoittaa mitään, sillä voidaan käyttää oletustyylejä.

  // Tyylimäärittelyt pöytätietokonenäytöille.
  @media (min-width: 768px) {
    float: left;
    width: 60%;
  }
}
```

Esimerkkikoodi 8. Kaksi eri lähestymistapaa sisältöelementin pöytätietokone- ja mobiilityylien kirjoittamiseen käyttäen mediakyselyä [55].

Esimerkistä huomataan, että jälkimmäisessä koodinpätkässä on säästetty kaksi riviä koodia, sillä pöytätietokonetyylejä ei tarvitse yliajaa eksplisiittisillä mobiilityyleillä. Vaikka tämä on varsin pelkistetty tapaus, se on riittävä havainnollistamaan, että mobile-first-ajattelulla on potentiaalia tuottaa tiiviimpää ja helppolukuisempaa CSS/SCSS-koodia. Vaikka onkin epätodennäköistä, että CSS-koodin pituus tulisi olemaan sivuston suorituskyvyn kannalta kaikkein suurin pullonkaula, on aina syytä huomioida, että mobiililaitteilla

on käytössään huomattavasti rajallisemmat määrät laiteresursseja ja siirtonopeutta verkon selaamiseen verrattuna kiinteään laajakaistaverkkoon kytkettyihin pöytätietokoneisiin. Mikäli mobiilityylit voidaan tarjota ensimmäisenä, luonnollisesti mobiiliselaimet lataavat ja jäsentävät tyylit nopeammin. Lisäksi on paremmat mahdollisuudet välttyä niin kutsutulta FOUC-ilmiöltä (flash of unstyled content, tyylittelemättömän sisällön välähdys), jolloin mobiilikäyttäjät saattavat nähdä hetkellisesti pöytätietokonenäkymän tyylit sivulatauksen alussa. [55; 56.]

WordPressin versiossa 4.4 RICG Responsive Images -lisäosan toiminnallisuus yhdistettiin osaksi WordPress-ydintä. Ohjelmistosulautuksen myötä teeman ja lisäosien kehittäjillä on ollut käytössään entistä paremmat työkalut ja tuki responsiivisten kuvien hallintaan WordPress-sivustoilla. Yhdistetty toiminnallisuus laajentaa olemassa olevaa wp_get_attachment_image-funktiota, jolloin artikkeleiden pää- ja sisältökuvien tulostettavaan HTML-merkintään sisällytetään oletuksena srcset- ja sizes-määritteet. Srcset sisältää kaikki käytettävissä olevien kuvakokojen osoitteet, kun taas sizes-määritteen tehtävänä on ohjeistaa selainta lataamaan oikeankokoinen kuva tietyllä selainikkunan leveydellä. Koska kuvien käyttö on kullekin verkkosivustolle yksilöllistä ja tapauskohtaista, teeman kehittäjän tulee huolehtia, että responsiivisilla kuvilla on käytössään tilanteeseen nähden mahdollisimman tarkat srcset- ja sizes-määritteiden arvot.

Kehittäjillä on käytössään useita erilaisia funktioita ja suodatinkoukkuja, joilla responsiivisten kuvien esittämisen logiikkaa voidaan muokata monenlaisiin konteksteihin. Vaikka WordPress generoikin oletuksena kuvan srcset-määritteen arvon tavalla, joka on useimpien sivustojen tarpeille riittävä, asianmukaisen sizes-määritteen arvon tuottaminen edellyttää tapauskohtaisen logiikan ohjelmointia. Esimerkkikoodissa 9 on Twentysixteen-teeman functions.php-tiedostossa käytetty suodatinfunktio, joka on kytketty wp_get_attachment_image_attributes-suodatinkoukkuun.

```
/**
 * Artikkeleiden pikkukuville lisätään mukautetut sizes-määritteiden arvot
 *
 * @since Twenty Sixteen 1.0
 *
 * @param array $attr Kuvan HTML-elementin määritteet.
 * @param WP_Post $attachment Kuvaliitteen WP_Post-olio.
 * @param array $size Teemaan rekisteröity kuvakoko tai taulukkoon tallennetut
 * pituus- ja korkeusarvot.
 * @return string Pikkukuvan HTML-elementin sizes-määritteen muokattu arvo.
 */
function twentysixteen_post_thumbnail_sizes_attr( $attr, $attachment, $size )
{
  if ( 'post-thumbnail' === $size ) {
```

```
    is_active_sidebar( 'sidebar-1' ) && $attr['sizes'] = '(max-width: 709px)
    85vw, (max-width: 909px) 67vw, (max-width: 984px) 60vw, (max-width:
    1362px) 62vw, 840px';

    !is_active_sidebar( 'sidebar-1' ) && $attr['sizes'] = '(max-width: 709px)
    85vw, (max-width: 909px) 67vw, (max-width: 1362px) 88vw, 1200px';
  }
  return $attr;
}
add_filter( 'wp_get_attachment_image_attributes', 'twentysixteen_post_thumb-
nail_sizes_attr', 10 , 3 );
```

Esimerkkikoodi 9. Suodatinkoukkuun kytketty funktio, jolla muokataan artikkelin pikkukuvalle tulostettavan sizes-määritteen arvoa [57; 58].

Mikäli kuvan koko vastaa teemaan rekisteröityä kuvakokoa ”post-thumbnail”, funktio tuottaa artikkelin pikkukuvalle asianmukaisen sizes-määritteen arvon sen mukaan, onko ”sidebar-1”-sivupalkki aktiivinen eli sisältääkö sivupalkki vimpaimia. Wp_get_attachment_image_attributes on siis sopiva suodatinkoukku, mikäli tarpeena on vaikuttaa artikkelin pääkuvan (engl. featured image) responsiivisuuteen. Artikkelin sisältökuvien tapauksessa on suositeltavaa käyttää wp_calculate_image_sizes-suodatinkoukkua sizes-määritteen arvon muokkaamiseen. Käytettäessä edellä mainittuja responsiivisten kuvien suodattimia on kuitenkin huomioitava, että kaikki teemaan rekisteröidyt rajatut kuvakoot, joiden kuvasuhteet poikkeavat alkuperäisestä kuvasta, jätetään pois srcset-määritteestä. [59; 60.]

Responsiivisten kuvien apufunktioita voidaan hyödyntää myös CSS-taustakuvien tapauksessa. Ei ole edullista pakottaa mobiililaitteita lataamaan taustakuvasta suuriresoluutioista versiota, joka on suunniteltu pöytäkonenäkymille. Tästä syystä olisi toivottavaa, että CSS-taustakuville voitaisiin HTML IMG -elementtien tavoin määrittää selaimille ohjeistuksia, minkäkokoinen versio kuvasta tulee ladata missäkin katselukontekstissa. Oletetaan tilanne, jossa ”news”-kategoriaan kuuluvan artikkelin pääkuva halutaan esittää artikkelisivun otsake-elementin taustakuvana. Esimerkkikoodissa 10 on esitetty funktio, jolla generoidaan automaattisesti responsiivisten taustakuvien tyylimäärittelyt pääkuvasta tallennetuille kuvako’oille.

```
add_action( 'wp_head', 'generate_responsive_background_image_styles');

function generate_responsive_background_image_styles() {
  if( in_category('news') ) {
    global $post;
    $image_id = get_post_thumbnail_id($post->ID);
    $img_srcset = wp_get_attachment_image_srcset( $image_id, 'full' );
    $sizes = explode( ", ", $img_srcset );
    asort($sizes, SORT_NATURAL | SORT_FLAG_CASE);
    $sizes = array_values($sizes);
    $css = '';
```

```
    $prev_size = '';
    foreach( $sizes as $size ) {
      $split = explode( " ", $size );
      if( !isset( $split[0], $split[1] ) )
          continue;

      $background_css = '.header {
          background-image: url(' . esc_url( $split[0] ) . ')
      }';

      if( !empty( $prev_size ) ) {
          $css .= '@media only screen and (min-width: ' . $prev_size . ') {';
          $css .= $background_css;
          $css .= "}\n";
      } else {
          $css .= $background_css;
      }

      $prev_size = str_replace( "w", "px", $split[1] );
    }

    $css = !empty( $css ) ? '<style>' . $css . '</style>' : '';
  }
}
```

Esimerkkikoodi 10. Wp_head-toimintokoukkuun kytketty funktio responsiivisten taustakuvien generointiin [61].

Funktio muuntaa wp_get_attachment_image_srcset-funktion palauttaman srcset-informaatiota sisältävän merkkijonon taulukoksi, järjestelee taulukon alkiot suuruusjärjestykseen, generoi foreach-silmukan sisällä jokaiselle kuvakokoa vastaavalle taustakuvalle asianmukaisen mediakyselylohkon ja tulostaa tyylimäärittelyt sivun head-osioon. Esimerkkikoodi 11 havainnollistaa, minkälaisia tyylisääntöjä funktio voi tulostaa.

```
<style>
.header {
  background-image: url(http://local.dev/wp-content/uploads/2016/04/image-
  300x151.png)
}
@media only screen and (min-width: 300px) {.header {
  background-image: url(http://local.dev/wp-content/uploads/2016/04/image-
  768x386.png)
}}
@media only screen and (min-width: 768px) {.header {
  background-image: url(http://local.dev/wp-content/uploads/2016/04/image-
  1024x515.png)
}}
@media only screen and (min-width: 1024px) {.header {
  background-image: url(http://local.dev/wp-content/uploads/2016/04/image.png)
}}
</style>
```

Esimerkkikoodi 11. Esimerkkikoodin 10 funktion mahdollinen HTML-tuloste.

4.2 WordPress-kehittämisen käytänteet

4.2.1 Teeman kehittämisen standardit

WordPress-ydin suorittaa taustalla monenlaisia operaatioita ja toimenpiteitä, joiden ansiosta teeman rakentamisen aloittaminen on lähtökohtaisesti nopeaa ja aloittelijaystävällistä. Useat taustaoperaatiot ovat perusrakenteeltaan kuitenkin melko suoraviivaisia ja edellyttävät toimiakseen tarkkaan määriteltyjen ehtojen täyttymisen. Esimerkkinä mainittakoon sivupohjahierarkian käyttämä logiikka oikean sivupohjan palauttamiseen sivunlatauksen aikana. Tarkastelemalla wp-includes-kansiossa sijaitsevaa template-loader.php-tiedostoa huomataan, että sivupohjan valinnan logiikka on toteutettu PHP if-elseif -konditionaaliketjulla. Konditionaaliketjussa artikkelisivuun viittaava koodirivi näyttää seuraavalta:

```
elseif ( is_single() && $template = get_single_template() ) :
```

18 rivin mittainen get_single_template-funktio taas olettaa, että teeman kansiosta on löydettävissä artikkelisivua koskeva sivupohjatiedosto, jonka nimi on muotoa {artikkelityypin sivupohjan nimi}.php, single-{artikkelityyppi}-{artikkelin nimi}.php, single-{artikkelityyppi}.php tai single.php [62]. Luonnollisesti teeman kehittäjän on siis huolehdittava artikkelisivua vastaavan sivupohjatiedoston asianmukaisesta nimeämisestä, jotta WordPress tämän tiedoston tunnistaisi. Sivupohjahierarkiaan nojautuva sivupohjatiedostojen nimeäminen on yksi esimerkki WordPress-teemakehityksen yleisesti hyväksytyistä käytänteistä. Luonnollisesti teemakehittäjän on myös huolehdittava, että yleisesti katsoen sovelluksen eri osaset on nimetty johdonmukaisesti ja itseään kuvailevasti. Hyvin ja kuvailevasti nimetty funktio voi säästää satoja sanoja dokumentaatiota. Seuraavassa on listattuna teemakehityksen nimeämiskäytänteitä:

- PHP-funktioiden ja -muuttujien sekä toiminto- ja suodatinkoukkujen nimeämisessä sovelletaan niin kutsuttua snake case -kirjoitustyyliä: sanat erotetaan toisistaan alaviivalla ja isoja alkukirjaimia ei käytetä. Niin kutsuttua camel case -kirjoitustyyliä tulee välttää. Jokaisen funktion sekä toiminto- ja suodatinkoukun nimessä on suositeltavaa käyttää etuliitteenä uniikkia tunnistetta, kuten teeman nimeä. Tällä pyritään välttämään mahdolliset yhteentörmäykset WordPress-ytimessä tai lisäosissa määriteltyjen funktioiden kanssa.

  ```
  function mytheme_some_func( $some_variable ) { [...] }
  ```

- PHP-luokkien nimissä esiintyvät sanat kirjoitetaan isoilla alkukirjaimilla ja erotetaan toisistaan alaviivalla. Kirjainlyhenteet kirjoitetaan isoilla kirjaimilla.

```
class Walker_Category extends Walker { [...] }
class WP_HTTP { [...] }
```

- PHP-vakioiden nimet kirjoitetaan isoilla kirjaimilla, ja sanat erotetaan toisistaan alaviivalla:

```
define( 'DOING_AJAX', true );
```

- Tiedostojen nimissä ei käytetä isoja kirjaimia, ja sanat erotetaan toisistaan väliviivalla. Mikäli tiedosto sisältää luokan koodin, tiedoston nimen tulee vastata luokan nimeä ja etuliitteeksi lisätään ”class-”. Luokan Walker_Category koodin sisältävä tiedosto nimettäisiin tällöin

```
class-walker-category.php
```

Kun tarkoituksena on kutsua tai suodattaa esimerkiksi artikkelityyppiin liittyvää dataa sivupohjatiedostoon esitettäväksi, kehittäjän on syytä muistaa, että tähän tarkoitukseen on todennäköisesti olemassa jokin valmis metodi, jolla toimenpide on suositeltavaa tehdä. Teeman sivupohjatiedostoissa ilmeneviä koodinpätkiä, joilla haetaan tietokannasta dataa ja tulostetaan se sivupohjatiedostoon, kutsutaan WordPress-kehittäjäyhteisössä sivupohjatageiksi (engl. template tag). Käytännössä sivupohjatagi koostuu kolmesta osasta: PHP:n avaus- ja sulkutageista, WordPress-funktiosta ja vaihtoehtoisista parametreista. Se, missä kontekstissa sivupohjafunktion kutsuminen on toimivaa ja jollain tapaa hyödyllistä, riippuu itse funktion luonteesta. Eräät funktiot saattavat edellyttää toimiakseen sen, että niitä kutsutaan WordPress-silmukan sisällä. Tällaiset funktiot edellyttävät globaalin $post-muuttujan olevan asetettuna. Luonnollisesti teeman kehittäjä ei siis voi kutsua esimerkiksi the_title-funktiota silmukan ulkopuolella. Sen sijaan otsikon hakuun tulisi tässä tapauksessa käyttää get_the_title-funktiota, edellyttäen että halutun artikkelin ID tai WP_Post-olio on käytettävissä. [63.]

Yleisesti katsoen WordPressin sivupohjafunktioiden käyttö on suositeltava tapa hakea ja tulostaa dataa. Sivupohjafunktioilla voidaan jakaa ja lohkottaa sisältöä selkeästi ymmärrettäviin osiin. Tämän lisäksi sivupohjafunktioilla on mahdollista helposti suodattaa ja muotoilla palautettavaa dataa eri tavoin käyttämällä funktiolle ominaisia parametreja. Esimerkiksi the_date- ja get_the_date-funktiot vastaanottavat parametreina PHP:n sisäisiä päivämäärän muotoilumerkkejä. Artikkelin julkaisuajankohta saadaan siis tulostettua suomalaisessa päiväysmuodossa kutsumalla silmukan sisällä funktiota the_date parametrilla ’j.n.Y’, jossa merkit j ja n vastaavat päivää ja kuukautta ilman etunollia. The_date-funktiota käytettäessä on kuitenkin syytä huomioida, että mikäli samalla sivulla tulostetaan useampia artikkeleita, joiden julkaisuajankohta on saman päivän ajalta, the_date-funktio tulostaa vain ensimmäisen artikkelin päivämäärän. Tällaisessa tapauksessa on suositeltavaa käyttää the_time-funktiota. [64.]

Usein sivustoon halutaan liittää kolmansien osapuolien skriptejä ja tyylejä erilaisia asiakaspään toiminnallisuuksia varten. Tavallisissa olosuhteissa kehittäjän tulee välttää skripti- ja tyylitiedostojen linkittämistä suoraan sivun HTML-merkintään header.php- tai footer.php-tiedostoon, sillä tämä saattaa johtaa suorituskykyongelmiin ja konflikteihin skriptien välillä. Tästä syystä kukin ulkoinen skripti- tai tyylitiedosto tulee liittää osaksi WordPress-sivun latausta käyttämällä tähän tarkoitettuja funktioita, wp_enqueue_script ja wp_enqueue_style. Yleinen tapa liittää useita JavaScript-tiedostoja WordPress-sivustoon on sisällyttää kaikkien skriptien lataus teeman functions.php-tiedostossa yhteen funktioon ja linkittää tämä funktio wp_enqueue_scripts-toimintakoukkuun. Esimerkkikoodi 12 havainnollistaa JavaScript-tiedostojen liittämisen osaksi teemaa käyttäen wp_enqueue_scripts-koukkuun kytkettyä funktiota.

```
function add_theme_scripts() {

  wp_enqueue_script( 'script', get_template_directory_uri() . '/js/script.js',
  array ( 'jquery' ), 1.1, true);

  if ( is_singular() && comments_open() && get_option( 'thread_comments' ) ) {
      wp_enqueue_script( 'comment-reply' );
  }
}
add_action( 'wp_enqueue_scripts', 'add_theme_scripts' );
```

Esimerkkikoodi 12. JavaScript-tiedostojen liittäminen teemaan käyttäen wp_enqueue_scripts-toimintakoukkua [65].

Kuten esimerkkikoodista huomaa, toimintafunktion käytöllä on tässä tilanteessa se etu, että eri skriptien lataamiselle voidaan asettaa ehtoja perustuen esimerkiksi WordPressin konditionaalifunktioihin tai tietokantaan tallennettuun asetusdataan. Vaikka Wp_enqueue_script- tai wp_enqueue_style-funktion polkuparametrissa olisi mahdollista käyttää apufunktiona joko get_template_directory_uri- tai get_stylesheet_uri-funktiota, on syytä muistaa, että get_template_directory_uri viittaa aina isäntäteeman kansion juureen. Kolmantena parametrina annettava taulukko sisältää kaikkien niiden rekisteröityjen skriptien nimet, joista kyseinen skripti on riippuvainen. Tätä hyödyntämällä WordPress siis huolehtii rekisteröityjen skriptitiedostojen oikeasta latausjärjestyksestä generoitavalle sivulle. Joidenkin osapuolten skriptien toiminta saattaa edellyttää skriptin linkittämisen sivun head-osioon, joidenkin taas ennen body-osion sulkutagia. Mikäli skripti tulee linkittää head-osioon, kehittäjän on huolehdittava, että wp_enqueue_script-funktion viides parametri on arvoltaan false.

WordPressin skriptien rekisteröinti- ja latausprosessissa jää varmasti monelta aloittelevalta teemakehittäjältä huomiotta mahdollisuus lähettää PHP-dataa JavaScript-koodin

käytettäväksi hyödyntämällä wp_localize_script-funktiota yhdessä wp_enqueue_script-funktion kanssa. Olkoon tilanne, jossa teeman js-alikansiossa istuva JavaScript-koodi tarvitsee informaatiota siitä, mikä on täysi URL-osoite kyseisen teeman kansioon. Esimerkkikoodissa 13 get_stylesheet_directory_uri-funktion palauttama merkkijono alustetaan assosiatiivisen taulukon, WPURLS, arvoksi. wp_localize_script-funktiossa vastaanottavan skriptin nimi on luonnollisesti sama kuin wp_enqueue_script-funktiolla ladatun skriptin nimi. URL-merkkijono tulee olemaan käytettävissä skriptitiedostossa newsletter-widget.js WPURLS-olion kentässä theme_path. [66.]

```
/* PHP koodia functions.php-tiedostossa */

wp_enqueue_script('newsletter-widget-js', get_template_directory_uri()
.'/js/newsletter-widget.js', array('jquery'), null, false);

wp_localize_script( 'newsletter-widget-js', 'WPURLS', array('theme_path' =>
get_stylesheet_directory_uri()) );


/* JavaScript-muuttuja newsletter-widget.js-tiedostossa */

var themePath = WPURLS.theme_path;
```

Esimerkkikoodi 13. PHP-koodista voidaan tuoda mielivaltaista dataa JavaScriptiin wp_localize_scriptin avulla.

Kehittäjän on myös syytä varmistaa, ettei skriptejä rekisteröidä tai ladata sivulle turhaan. WordPressiin on nimittäin tähän päivään mennessä sisällytetty jo useita kymmeniä verkkokehityksessä usein käytettyjä JavaScripteja. Näistä tunnettuina esimerkkeinä mainittakoon jQuery-kirjasto ja sitä hyödyntävä jQuery UI -käyttöliittymäkokoelma, tinyMCE-editori, Prototype JavaScipt -sovelluskehys sekä Underscore.js- ja Backbone.js -kirjastot. Koska WordPressiin valmiiksi asennetut JavaScriptit ovat rekisteröityjä skriptejä, niitä ei siis tarvitse erikseen ladata sivulle wp_enqueue_script-funktiolla. Esimerkiksi, jos kehittäjän kirjoittama skriptitiedosto on riippuvainen jQuerystä, määritettäessä se wp_enqueue_script-funktion parametreissa vaadittujen skriptien joukkoon WordPress lataa jQueryn sivulle automaattisesti ennen kehittäjän skriptitiedostoa. [65; 67.]

Hyviin käytänteisiin liittyy olennaisesti modulaarinen hierarkia. Lähtökohtana on, että verkkosovelluksen erilaiset ominaisuudet on jaettu omiin loogisiin komponentteihin ja olioihin, jotka toimivat toisistaan riippumattomasti. Ihannetapauksessa erilaiset sivustotoiminnallisuudet ovat suljettuja komponentteja, jotka toimivat kuten lisäosat: ne voidaan kytkeä sivustoon ja siitä pois ilman, että sivuston toiminta häiriintyisi merkittävästi. Vaikka teemakehityksen näkökulmasta olisi täysin mahdollista, että kaikki toiminta- ja suodatinkoukut, PHP-oliot ja -funktiot, vimpainten ja mukautettujen artikkelityyppien logiikka ja

teema-asetukset kirjoitettaisiin suoraan teeman functions.php-tiedostoon, tämä menettelytapa on ristiriidassa modulaarisen ajattelun kanssa.

Modulaarisen verkkosovelluksen ylläpidon kannalta on välttämätöntä, että sovelluksen kansiorakenne on siisti ja looginen. Style.css-, functions.php- ja geneerisimmät sivupohjatiedostot sijaitsevat teemakansion juuressa. Erilaiset pienemmät ja uudelleenkäytettävät sivupohjapalaset sijoitetaan alikansioon template-parts. Verkkoaineistot, kuten kuvat, grafiikat sekä JavaScript- ja tyylitiedostot, on tapana sijoittaa kansioon assets ja edelleen asianmukaisiin alikansioihin tiedostoformaatin perusteella, kuten img, css ja js. Projektista tai kehittäjätiimin käytänteistä riippuen erilaiset teeman ydintoiminnallisuuksiin liittyvät PHP-funktiot ja -oliot voidaan sijoittaa esimerkiksi alikansioihin inc (includes) tai lib (library). Usean kehittäjän projekteissa on edullista, että tietyntyyppiset ja tiettyä tehtävää hoitavat PHP-funktiot kerätään asianmukaisesti nimettyihin tiedostoihin, kuten actions.php, filters.php tai shortcodes.php. Yleiskäyttöisiä, globaaleja apufunktioita kootaan yleensä tiedostoon utils.php, helpers.php tai vastaavaan. Teeman functions.php ei siis tarvitse itsessään sisältää lainkaan ydintoiminnallisuutta, vaan se voi toimia enemmänkin kääreenä, johon alikansioihin sijoitetut, pienempiin kokonaisuuksiin jaetut komponentit sisällytetään include-lauseilla.

Erityisen tärkeä WordPress-kehittäjän taito on hahmottaa erilaisten komponenttien ja sivustotoiminnallisuuksien kontekstisidonnaisuus. Tähän liittyvät WordPress-kehitysprojekteissa jatkuvasti vastaantulevat kysymykset siitä, onko komponentti järkevämpää toteuttaa sivupohjana vai vimpaimena tai kirjoitetaanko tietty olio teeman functions.php-tiedostoon vai rakennetaanko sille oma lisäosa. Lähtökohtaisesti, mikäli kyseessä on toiminnallisuus, johon liittyvä data ja logiikka halutaan säilyttää sivustolla riippumatta käytössä olevasta teemasta, tämä komponentti tulee kirjoittaa omaksi lisäosakseen. Mukautetut artikkelityypit (custom post type) ovat hyvä esimerkki komponenteista, joita harvemmin on suositeltavaa sisällyttää osaksi teemaa, sillä teemaa vaihdettaessa niihin liittyvä toiminnallisuus ja sisältö luonnollisesti menetetään. Kun sivuilla halutaan esittää sisältöä tai dataa, joka ei ole millään tavalla sidonnainen tietyn sivupohjan sisältöön tai esityslogiikkaan, on järkevää rakentaa komponentti vimpaimeksi. Sen sijaan, jos komponentin rakenne tai data vaihtelee sivupohjasta toiseen, on syytä harkita komponentin toteuttamista sivupohjatiedostona.

Vimpaimet ovat sopiva tapa esittää vaihtoehtoista sisältöä, jota voidaan tuoda mielivaltaiseen paikkaan mille tahansa sivulle. Vimpaimien toiminta muistuttaa lisäosia siinä

mielessä, että ne täydentävät sivun varsinaista sisältöä ja ne voidaan ottaa käyttöön tai poistaa vaikuttamatta muuhun sisältöön. Vimpaimet kuitenkin poikkeavat WordPress-lisäosista siinä, että lisäosien tarkoitus on lähtökohtaisesti laajentaa sivuston toiminnallisuutta, kun taas vimpaimien toiminta painottuu enemmänkin puhtaasti tietyntyyppisen sisällön tai datan esittämiseen. Tilanteesta ja sivuston omistajan tavoitteista riippuen vimpaimet voivat olla sivupalkkiin tai muuhun vimpainalueeseen kiinnitettäviä, lisukesisältöä esittäviä palikoita tai koko sivu voi olla koostettu yksinomaan vimpaimista ja niiden tulostamasta sisällöstä.

Vimpainrajapinnan hyödyntäminen on hyvä käytäntö siinäkin mielessä, että erilaisten sivustotoiminnallisuuksien rakentaminen vimpaimiksi kannustaa modulaariseen ajatteluun ja olio-ohjelmointiin. Vimpain itsessään on PHP-olio, jonka tehtävä on yksinkertainen: tulostaa WordPressin ohjausnäkymään asetuslomake, tallentaa käyttäjän syöttämien asetusten data tietokantaan wp_options-tauluun ja tulostaa sivulle HTML-sisältöä annettujen asetusten perusteella. Räätälöidyn vimpaimen tulee laajentaa WordPress-luokkaa WP_Widget esimerkkikoodissa 14 näkyvällä tavalla:

```
class My_Custom_Widget extends WP_Widget {

    public function __construct() {
      parent::__construct(
        'my_custom_widget',
        esc_html__( 'My Custom Widget', 'text_domain' ),
        array( 'description' => esc_html__( 'Tämän vimpaimen avulla määritel-
        lään kelluvat some-ikonit sivulle', 'text_domain' )
      );

      add_action( 'widgets_init', function() {
        register_widget( 'My_Custom_Widget' );
      });
    }

    public function widget( $args, $instance ) {
      // tulostaa vimpaimen sisällön sivulle
    }

    public function form( $instance ) {
      // tulostaa vimpaimen asetuslomakkeen ohjausnäkymään
    }

    public function update( $new_instance, $old_instance ) {
      // prosessoi asetuskenttien datan ennen tietokantaan tallennusta
    }

}
```

Esimerkkikoodi 14. Räätälöity vimpainluokka, joka laajentaa WP_Widget-luokkaa [68].

Tässä esimerkkitapauksessa vimpainluokka sijoitetaan tiedostoon my-custom-widget.php, esimerkiksi polkuun src/Widgets. Luokan konstruktorissa kutsutaan isäntäluokan konstruktoria, jolle syötetään vimpaimen yksilöivä tunniste sekä vimpaimen nimi ja seliteteksti, jotka tulostetaan ohjausnäkymän vimpainvalikossa vimpaimen tiedoissa. Luokan konstruktorissa myös rekisteröidään kyseinen vimpain widgets_init-koukun kautta. Vimpaimen rekisteröinti on välttämätöntä, jotta vimpain olisi nähtävissä ohjausnäkymän vimpainasetusnäkymässä. $args on assosiatiivinen taulukko, joka sisältää teemaan rekisteröityyn vimpainalueeseen määritetyt ’before_title’- ja ’after_title’-, ’before_widget’- ja ’after_widget’-kenttien arvot. Mikäli vimpainalueen rekisteröintivaiheessa $args-taulukko on määritetty, tämä taulukko välitetään jokaisen vimpainalueeseen asetetun vimpaimen widget-funktion ensimmäiseksi parametriksi. Vimpainluokan sisällä $args-taulukkoa käytetään pääasiassa vaihtoehtoisten HTML-kääreiden avaus- ja sulkutagien tulostamiseen vimpaimen otsikon ja itse vimpainelementin ympärille. [68; 69.]

$instance on tyypiltään taulukko, joka sisältää vimpainluokan instanssin asetuskenttien arvot, esimerkiksi otsikon tapauksessa $instance[’title’]. Mikäli kyseessä on vimpainalueeseen kiinnitettävä vimpain, $instance-taulukon alkioiden arvot tulevat vimpaimen asetuslomakkeen kentistä, jotka täytetään ohjausnäkymässä vimpainasetussivulla. Koska kukin vimpain käsitellään omana instanssinaan, sama vimpain on mahdollista asettaa ohjausnäkymässä useita kertoja samaan vimpainalueeseen, jolloin kullakin kiinnitetyllä vimpain-instanssilla on omat yksilölliset asetusarvot. Instanssin asetuskenttien arvot tallennetaan tietokantaan vimpaimen update-funktiossa, joka palauttaa käyttäjän syöttämät uudet asetukset sisältävän taulukon $new_instance. Itse vimpainelementin tulostamisesta sivulle huolehtii vimpainluokan widget-funktio, joka käytännössä voi tulostaa täysin mielivaltaista HTML-koodia ja PHP-dataa. Vimpaimen instanssin asetuskenttien arvoja käytetään widget-funktiossa moninaisin tavoin, esimerkiksi tietokantakyselyiden suodattamiseen, jotta tietynlaisia artikkelilistauksia saadaan palautettua mielekkäällä tavalla.

4.2.2 Ohjelmoinnin käytänteet

Epäilemättä ohjelmointi on huolellisuutta ja tarkkuutta edellyttävää työtä. Inhimillisestä huolimattomuudesta aiheutuneet koodivirheet eli bugit ja vikojenetsintä eli debuggaaminen ovat kehittäjän arkipäivää. Hyvien ohjelmointikäytänteiden noudattamisella pyritään ennaltaehkäisemään kaikkein yleisimpiä ohjelmointivirheitä ja säilyttämään ohjelmakoodi mahdollisimman helppolukuisena ja ylläpidettävänä. Koodin tulee näyttää siltä, kuin sen olisi kirjoittanut yksi ja sama henkilö. Ideaalitapauksessa kuka tahansa kykenee

ymmärtämään ohjelmakoodin ja tekemään siihen muokkauksia riippumatta siitä, kuka tai ketkä koodin alun perin kirjoittivat tai kirjoitettiinko koodi viisi päivää vai viisi vuotta sitten.

Datan tulostamisessa sovellettava heitto- ja lainausmerkkien oikeaoppinen käyttö sekä käsittelyn esto (engl. escaping) aiheuttavat monille PHP-ohjelmoinnin aloittelijoille hankaluuksia. Tulostettaessa sivulle HTML-merkistöä nyrkkisääntönä voidaan pitää seuraavaa:

- Mikäli tulostettava merkkijono ei sisällä suoritettavaa PHP-koodia tai -muuttujia, merkkijonon uloimpina erotinmerkkeinä käytetään heittomerkkejä. HTML-tagien määritteiden staattiset arvot sisällytetään tavallisesti lainausmerkkien sisään:

```
echo '<a href="/static/uutiset" title="Uutiset">Uutiset</a>';
```

- Jos merkkijono sisältää suoritettavaa PHP-koodia tai -muuttujia, merkkijonon uloimpina erotinmerkkeinä käytetään lainausmerkkejä. Tällöin merkkijonon sisällä vastaantulevat muuttujat suoritetaan eikä tulkita sellaisenaan. HTML-määritteisiin asetetut muuttujat sisällytetään heittomerkkien sisään:

```
echo "<a href='$link' title='$linktitle'>$linkname</a>";
```

Tulostettavissa merkkijonoissa on tarkoitus pitkälti siis välttää käyttämästä kenoviivaa käsittelyn estoon, jotta koodi säilyisi siistinä ja luettavana. Mikäli kyseessä on funktio, jonka tarkoituksena on tulostaa HTML-merkintää sivupohjatiedostoon, vaihtoehtoisesti tulostettavat HTML-lohkot voidaan erottaa PHP-koodista PHP:n avaus- ja sulkutageilla, jolloin funktioon kirjoitettu HTML-merkintä tulostuu sivulle sellaisenaan. Tässä lähestymistavassa on syytä kuitenkin huomioida koodin selkeä ja looginen rivittäminen ja sisentäminen. Esimekkikoodi 15 esittää HTML-merkintää tulostavan funktion koodin johdonmukaisen sisentämisen.

```
function foo() {
  ?>
    <div>
      <?php
        echo bar(
          $baz,
          $bat
        );
      ?>
    </div>
  <?php
}
```

Esimerkkikoodi 15. PHP-avaus- ja -sulkutagit sisennetään samoille sarakkeille [70].

Koodin johdonmukainen rivittäminen ja sisentäminen ei ole itsestäänselvyys. Melko yksinkertaisen rutiinin soveltamisella voi olla merkittävä vaikutus lähdekoodin lukukelpoisuuteen. Esimerkkinä mainittakoon assosiatiiviset taulukot, joiden tapauksessa on suositeltavaa erotella taulukon alkiot omille riveilleen, kuten esimerkkikoodissa 16.

```
$query = new WP_Query( array(
  [tab]'post_type'   => 'page',
  [tab]'post_author' => 123,
  [tab]'post_status' => 'publish',
) );
```

Esimerkkikoodi 16. Mikäli taulukon alkioita on enemmän kuin yksi, kukin alkio rivitetään [70].

Edeltävässä esimerkkikoodissa on syytä kiinnittää huomiota myös sijoitusoperaattorien linjaukseen. Useimmissa tapauksissa sijoitusoperaattorien linjauksella samalle sarakkeelle on luettavuutta edistävä vaikutus, ja tämä onkin ainoa tapaus, jossa välilyöntien käyttö sisentämiseen on perusteltua. Muissa tapauksissa tulee lähes aina käyttää sarkainta. Toinen huomion arvoinen yksityiskohta esimerkkikoodissa 16 on pilkun lisäys taulukon viimeisen alkion loppuun. Näin tehdään puhtaasti käytännön kannalta, sillä uusien elementtien lisääminen ja elementtien järjestyksen muokkaaminen on näin vähemmän virhealtista. [70.]

PHP-datan ja HTML-merkinnän tulostamiseen on useissa tapauksissa hyväksyttävää käyttää echo-lausetta tai kirjoittaa HTML-merkintää suoraan sellaisenaan tulostavaan funktioon. Olennainen heikkous näissä lähestymistavoissa kuitenkin on se, että sovelluksen esityskerros tulee sekoitetuksi suorituslogiikan kanssa, eikä tämä juurikaan palvele modulaarisuutta. Lähtökohtaisesti tulisi pyrkiä siihen, että sovellusnäkymään liittyvät komponentit pidetään sovelluslogiikasta ja -toiminnoista erillään. Lyhyesti ilmaistuna, funktioiden kuuluu pääosin kysellä, käsitellä ja palauttaa dataa, kun taas sivupohjien kuuluu ainoastaan vastaanottaa PHP:n prosessoima data ja esittää se HTML-merkinnässä. On erityisesti myös huomioitava, että mikäli vastaan tulee tilanteita, joissa mielivaltaista HTML-merkintää tai sivupohjan sisältö tulee tallentaa muuttujaan ja palauttaa toisen komponentin prosessoitavaksi, tavanomainen tulostaminen esimerkiksi echo-lauseella tai get_template_part-funktiolla ei ole tässä tapauksessa kovinkaan mielekästä. Tulostettavan HTML-merkinnän tallennukseen ja palauttamiseen on mahdollista käyttää PHP:n tulosteen puskurointimekanismia (engl. output buffering control).

Oletetaan tilanne, jossa halutaan hakea tietty lukumäärä artikkeliolioita tietokannasta ja esittää olioiden dataa artikkelilistauksessa lyhytkoodin kautta mielivaltaisessa paikassa

artikkelisivun sisällössä. Esimerkkikoodi 17 havainnollistaa erään lähestymistavan tähän ongelmaan.

```
add_shortcode(
  'recent_posts',
  'mytheme_get_recent_posts_shortcode'
);

function mytheme_get_recent_posts_shortcode() {

  $posts = new WP_Query( array('post_type' => 'post', 'posts_per_page' => 5)
);
  $output = '';

  if ( $posts->have_posts() ) {
    $posts = $posts->get_posts();
    ob_start();
    ?>

    <div class="popular-posts-list">
      <h2 class="shortcode-title">Uusimmat</h2>

    <?php
    foreach( $posts as $post ) {
      setup_postdata( $GLOBALS['post'] =& $post );
      include( locate_template('includes/post-template.php') );
      wp_reset_postdata();
    }
    ?>

    </div>

    <?php
    $output = ob_get_clean();
  }

  return $output;
}
```

Esimerkkikoodi 17. Lyhytkoodin funktio, joka käyttää tulosteen puskurointia HTML-tulosteen tallentamiseen ja palauttamiseen.

Esimerkkikoodissa ob_start-funktiolla avataan tulostepuskuri, minkä jälkeen kaikki tulostuva data tallennetaan sisäiseen puskuriin. Avattu puskuri tulee toki myös sulkea lopuksi. ob_get_clean-funktio suorittaa samanaikaisesti syötteen palautuksen, puskurin sisällön tyhjennyksen ja puskurin sulkemisen. Jokaisen artikkeliolion data tulostetaan sivupohjatiedostossa post-template.php, joka pyritään pitämään mahdollisimman puhtaana kaikesta PHP-koodista. Tärkeä huomio edellä mainitussa esimerkkitapauksessa on se, että mikäli tulosteen puskurointia ei käytettäisi ja lyhytkoodin funktio tulostaisi datan suoraan, artikkelilistaus ilmestyisi aina artikkelisivun sisällön ylälaitaan. [71; 72.]

JavaScript-ohjelmoinnista on tullut keskeinen osa modernien WordPress-sivustojen ja -lisäosien kehitystä. Tämän vuoksi ei liene yllättävää, että siinä, missä PHP-kielenkin

osalta, myös JavaScriptin kirjoittamisen standardisointi on ylläpidollinen välttämättömyys. Koodin kirjoittamisen osalta kehittäjän ei ole syytä arastella rivinvaihtoja reilujen rivivälien aikaansaamiseksi. Yleensä tämä vain auttaa koodin osien ja kulun hahmottamista, ja skriptit yhdistetään ja minifioidaan joka tapauksessa tehtävänkäsittelijän (esimerkiksi Grunt tai Gulp) tai muun vastaavan sovelluksen koontiprosessissa ennen tuotantoonvientiä. Tarpeettoman pitkiä rivejä on syytä välttää; viitteellinen suositus on pitää koodirivit alle 80–100 merkin pituisina. Sisennyksiin käytetään ainoastaan sarkainta, jonka pituus yleensä vastaa joko kahta tai neljää välilyöntiä. Konditionaaliketjujen ja silmukoiden tulee aina käyttää aaltosulkeita ja jakautua usealle riville. On myös hyvä tapa jättää JavaSrcipt-tiedoston loppuun yksi tyhjä rivi. Vaikka tämän tekemättä jättämisestä harvoin varsinaisesti seuraa ongelmia, saatetaan sillä välttää tiedostojen yhdistämisprosessissa tapahtuvia sivuvaikutuksia.

JavaScript-ohjelmoijan on erityisesti huolehdittava, että skriptit eivät saastuta nimiavaruutta globaaleilla muuttujilla. Globaalien muuttujien käyttö nähdään monissa konteksteissa paheksuttavana käytänteenä, koska luonnollisesti ulkopuolisten, täysin eri tarkoituksia palvelevien skriptien ja komponenttien ei haluta tarkoituksettomasti muokkaavan toistensa dataa. Yleinen tapa välttää globaalien muuttujien käyttöä on hyödyntää JavaScript-sulkeumia (engl. closure) ja niin kutsuttua moduulimallia. Periaatteena on sulkea skriptin varsinainen koodi moduulin sisään siten, että skriptin kontekstista katsottuna käytetään ainoastaan lokaaleja muuttujia ja funktioita. Esimerkkikoodissa 18 on esitetty yksinkertainen JavaScript-moduuli.

```
var counter = (function() {

  var privateCounter = 0;

  function changeBy(val) {
    privateCounter += val;
  }

  return {
    increment: function() {
      changeBy(1);
    },
    decrement: function() {
      changeBy(-1);
    },
    value: function() {
      return privateCounter;
    }
  };

})();
```

Esimerkkikoodi 18. JavaScript-moduuli, joka käyttää yksityisiä muuttujia ja funktioita [73].

Esimerkkikoodissa muuttujaan counter on tallennettu anonyymi funktio, joka suoritetaan välittömästi, kun se on määritelty. Tämä funktio sisältää suljetun leksikaalisen näkyvyysalueen, joka piilottaa muuttujan privateCounter ja funktion changeBy globaalilta näkyvyysalueelta. Yksityisiin muuttujiin ja funktioihin ei siis voi viitata suoraan globaalissa kontekstissa. Sen sijaan niitä voidaan käyttää julkisilla sulkeumafunktiolla, jotka palautetaan anonyymin kääreen sisällä isäntäfunktiosta. Esimerkkikoodissa sulkeumafunktiot jakavat saman leksikaalisen näkyvyysalueen ja voivat viitata alueella määriteltyihin lokaaleihin muuttujiin ja funktioihin, vaikka isäntäfunktio on jo käytännössä suoritettu. JavaScriptissä leksikaaliset näkyvyysalueet ja niiden sisälle määritellyt muuttujat ja funktiot tallennetaan muistiin, ja niihin voidaan viitata myöhemmin sulkevan funktion suorituksen jälkeen. Jos koodissa ajettaisiin kolme kertaa counter.increment ja tämän jälkeen tulostettaisiin lokiin counter.value, näkyisi lokissa arvo 3. JavaScriptissä moduulit ovat siis eräänlainen tapa imitoida yksityisiä kenttiä ja metodeja käyttäviä luokkia. [73.]

Mikäli kehittäjä kirjoittaa skriptit käyttäen jQuery-kirjastoa, on syytä huomioida, että WordPress-asennukseen sisällytetty jQuery on asetettu oletuksena niin kutsuttuun noConflict-tilaan. Tämä tarkoittaa käytännössä sitä, että tavallisesti jQuery-funktion kutsuun käytettävää lyhennettä ”$” ei voi käyttää. Tämä on mahdollista sivuuttaa syöttämällä jQuery-olio välittömästi kutsuttavalle anonyymille funktiolle, lyhyemmin IIFE-funktiolle (engl. immediately invoked function expression), esimerkkikoodin 19 esittämällä tavalla.

```
(function( $ ) {
  // jQuery koodi
})( jQuery );
```

Esimerkkikoodi 19. jQuery-olio nimetään anonyymille IIFE-funktiolle syötetyllä parametrilla [74].

Vaikka jQueryllä DOM-elementtien valitseminen on tavanomaisissa olosuhteissa verrattain nopeaa, samoihin elementteihin kohdistuvat toistuvat kyselyt ovat luonnollisesti haaskausta. Edullisempaa on tallentaa erilaisten kyselyiden ja funktiokutsujen arvoja muuttujiin aina, kun mahdollista. Esimerkiksi jos lomakkeen lähetyspainikkeeseen viitataan skriptissä useita kertoja, viittaus painikkeen DOM-olioon on hyvä tallentaa muuttujaan:

```
var $formSubmit = $('.newsletter-form input[type="submit"]');
```

Mikäli painikkeen DOM-olioon tulee myöhemmin kiinnittää esimerkiksi klikkaustapahtuma, voidaan olioon viitata muuttujan kautta seuraavasti:

```
$formSubmit.on('click', function() { ... });
```

Edeltävä esimerkki on vain yksi tapaus monien joukossa, milloin datan tallentaminen väliaikaiseen muistiin mielletään hyväksi käytänteeksi suorituskyvyn näkökulmasta. Esimerkiksi jos iteroidaan for-silmukassa hyvin suurta joukkoa JavaScript-olioita, on suositeltavaa esimerkkikoodin 20 esittämällä tavalla tallentaa viittaus silmukan maksimiarvosta muuttujaan sen sijaan, että arvo laskettaisiin joka kierroksella olioiden lukumäärän perusteella.

```
var i, objsLength;

objsLength = objs.length;

for (i = 0; i < objsLength; i++) {
    // Do stuff
}
```

Esimerkkikoodi 20. JavaScript for-silmukan maksimiarvo tallennettuna muuttujaan.

Silmukoiden käytössä on lisäksi syytä muistaa, että jQuery-silmukoiden käyttöä tulee välttää sellaisissa tapauksissa, joissa iteroidaan muita JavaScript-datatyyppejä kuin jQuery-olioita. [74.]

5 Verkkosivuston toteutus ja käyttöönotto

Opinnäytetyön projekti suoritettiin full-stack-luonteisena verkkokehitystyönä helsinkiläiselle mainostoimistolle. Projektin tavoitteena oli rakentaa yrityksen brändiä julkaisemalla nykyaikaistetut kotisivut. Projekti sisälsi räätälöidyn WordPress-teeman toteutuksen yrityksen visuaalisen suunnittelutiimin työstämien layoutien pohjalta. Sisäisesti tuotettuja verkkoaineistoja (kuvat, grafiikka, fontit ja niin edelleen) lukuun ottamatta, sivusto rakennettiin kokonaisuudessaan itsenäisesti yrityksen toimitusjohtajan sekä digitaalisen liiketoiminnan johtajan ohjeistuksen mukaan. Kehitystyössä pyrittiin huomioimaan, että projektin lopputuotteena syntyvä sivusto täyttäisi seuraavat kriteerit:

- Sivuston ulkoasu on responsiivinen ja sivustotoiminnallisuuksien rakentamisessa on huomioitu myös mobiilikäyttäjät. Sivusto toimii esteettömästi ja näyttää samalta kaikilla tunnetuilla moderneilla selaimilla (Chrome, Safari, Firefox, Edge/Internet Explorer +10).

- Sivuston asiakaspäässä ilmentyy useita modernin verkkokäyttöliittymän suunnittelumalleja, kuten koko selainikkunan täyttävä landing-alue HTML5-taustavideolla, täysileveisiin tieto-osioihin ja taustakuviin jaettu, alaspäin vieritettävä etusivu sekä CSS3-tekniikalla toteutetut värisuodattimet ja hover-animaatiot.
- Sivuston sisältö ja data on dynaamista; suurin osa asiakaspäässä esitettävästä informaatiosta on lähtöisin WordPress-ohjausnäkymän asetuskentistä ja editoreista. Sisällön hallinta ja muokkaus toimii käyttäjäystävällisesti.

Vaikka yritys otti pääasiallisen vastuun sivuston visuaalisten ja toiminnallisten suuntaviivojen määrittelystä, projektin alkuvaiheen suunnittelussa toimittiin tiiviissä yhteistyössä yrityksen henkilöstön kanssa. Itse sivuston kehittämisessä käytettävien metodien ja työkalujen suhteen annettiin suhteellisen vapaat kädet, edellyttäen että ohjeistuksessa annettavat visuaaliset ja tekniset vaatimukset täyttyvät lopputuloksessa. Kehitystyössä sovellettiin HTML5-, CSS3-, JavaScript-/jQuery- ja PHP-tekniikoita. Kehitystyössä pyrittiin lisäksi rakentamaan sivusto kunnioittaen WordPress-teemakehityksen hyviä käytänteitä.

Uudistettujen kotisivujen rakenne jätettiin verrattain yksinkertaiseksi: itse suomen ja englannin kielillä toteutetun etusivun lisäksi sivusto sisälsi ainoastaan yrityksen tarjoamien palvelujen esittelysivut ja perinteiset blogiartikkelisivut. Yhteydenotto- ja uutiskirjelomake ovat sivuston ainoat komponentit, jotka käsittelevät käyttäjän syöttämää dataa. Näissäkin komponenteissa toimintaperiaate on yksinkertainen: lomakkeet on toteutettu Contact Form 7 -lisäosalla, joka lähettää palvelimelta sähköpostin ohjausnäkymän asetuksissa määriteltyyn osoitteeseen. Projektin pienuuden ja yksinkertaisen luonteen vuoksi sivusto soveltui varsin hyvin kehitettäväksi täysin lokaalissa ympäristössä. Lokaali kehittäminen ja testaus suoritettiin PC:llä (Windows 7), johon oli asennettu WAMP-sovelluspino.

Tehokkaan kommunikaation ja riskien minimoinnin näkökulmasta katsottiin kuitenkin tarpeelliseksi, että kehitettävä sivusto olisi myös tarkastettavissa ja testattavissa erillisellä kehityspalvelimella. Kehityspalvelimena toimi henkilökohtaisen Amazon AWS -profiilin hallintakonsolin kautta käyttöönotettu t2.micro-tyypin EC2-instanssi. Palvelininstanssin konfigurointiin käytettiin Bitnamin sertifioimaa AMI-levykuvaa, johon sisältyi esiasennettu Linux-käyttöjärjestelmän Ubuntu 14.04.3 -jakeluversio ja WordPress 4.7.3. Tiedostojen siirtoon lokaalista kehitysympäristöstä testipalvelimelle käytettiin WinSCP-asiakasohjelman kautta muodostettua SSH RSA -avainparilla suojattua FTP-yhteyttä. Kehityssivusto oli nähtävissä ja testattavissa julkisen IP-osoitteen takana. Projektin versionhallintaan käytettiin Gitiä, ja sivuston lähdekoodi tallennettiin henkilökohtaiseen Github-säilöön.

Etusivun osioiden sisällönhallintakomponenttien rakentamisen keskeisessä osassa toimi WordPressin teemamukautinrajapinta (engl. theme customize API). Teemamukautin on oliopohjainen tapa määrittää erilaisia asetuspaneeleita ja -kenttiä sivuston ulkoasun ja sisältöelementtien muokkausta varten. Teemamukauttimen käytön etuna verrattuna tavanomaisiin ohjausnäkymän asetussivuihin on se, että muutettavaa sisältöä ja tyylejä on mahdollista esikatsella sivulla ennen tallentamista. Teemamukauttimen kenttien kautta syötettävä mielivaltainen data voidaan tallentaa joko tavanomaisina sivustoasetuksina tai teemakohtaisina asetuksina. Nämä eroavat käytännössä siinä, että kukin yksittäinen sivustoasetuskenttä tallennetaan tietokannan options-tauluun omana tietueenaan, kun taas kaikki teemakohtaiset asetukset tallennetaan options-taulun tietueeseen theme_mods_[teeman nimi] serialisoituna taulukkona. Luonnollisesti tämä tarkoittaa sitä, että teemakohtaisiin asetuksiin tallennettu data menetetään teemaa vaihdettaessa.

Teemamukautin koostuu neljäntyyppisistä olioista: paneeleista, osioista, asetuksista ja kontrolleista. Asetuksella kytketään tietokantaan tallennettava asetusdata kontrolliin, jolla muokataan kyseistä dataa. Kontrolli on siis käyttöliittymäkomponentti, jonka tyyppi on riippuvainen itse manipuloitavan datan tyypistä. Mikäli kyseessä on merkkijonodata, luonnollisesti kontrollin tyyppi olisi tällöin yhden tai useamman rivin tekstikenttä. Osiot ovat nimettyjä alueita, joiden sisällä halutut asetuskentät tulostetaan. Mikäli eri osioita halutaan edelleen jaotella hierarkkisesti omiin ryhmiin, voidaan tähän tarkoitukseen käyttää teemamukauttimen paneeleita. Jokaista teemamukautinrajapinnan kautta rekisteröityä oliota hallinnoi luokka WP_Customize_Manager, joka toimii tehtaana rekisteröitäville olioille. Teemaan rekisteröitävät mukautinoliot on määritettävä käyttäen customize_register-toimintakoukkuun kytkettyä funktiota. Projektissa mukautinoliot määritettiin functions.php-tiedostossa toimintafunktion sisällä esimerkkikoodin 21 esittämällä tavalla.

```
add_action( 'customize_register', 'smoy_customize_register' );

function smoy_customize_register( $wp_customize ) {

  // Osion määritys
  $wp_customize->add_section( str 'osion_id', array $asetukset );

  // Asetuksen määritys
  $wp_customize->add_setting( str 'asetuksen_id', array $asetukset );

  // Kontrollin määritys
  $wp_customize->add_control( str 'kontrollin_id', array $asetukset );

}
```

Esimerkkikoodi 21. Mukautinoliot rekisteröidään teemassa käyttämällä customize_register toimintakoukkua.

Toimintafunktiolle syötettävällä parametrilla $wp_customize viitataan WP_Customize_Manager-luokan instanssiin. Kutsumalla luokan add_-funktiota voidaan lisätä haluttu mukautinolio teemaan. Mukautinkenttään tallennettua dataa voidaan hakea kutsumalla funktiota get_theme_mod('asetuksen_id'). Esimerkkikoodissa 22 on esitetty katkelma projektin functions.php-tiedostosta. Katkelma havainnollistaa, kuinka teeman mukauttimeen lisättiin asetuskenttä, jolla vaihdetaan etusivun taustakuvaosion kuvatiedosto.

```
$wp_customize->add_section( 'smoy_bg_imgs', array(
  'title' => __( 'Front page background images', 'smoy' ),
  'description' => __( '...', 'smoy' ),
  'capability' => 'edit_theme_options'
));

$wp_customize->add_setting('smoy_about_us_bg_img', array(
   'type' => 'theme_mod',
   'capability' => 'edit_theme_options',
   'sanitize_callback' => 'absint'
));

$wp_customize->add_control( new WP_Customize_Cropped_Image_Control(
  $wp_customize, 'smoy_about_us_bg_img', array(
    'section'     => 'smoy_bg_imgs',
    'label'       => __('About us background image', 'smoy'),
    'description' => __('...', 'smoy'),
    'flex_width'  => false,
    'flex_height' => false,
    'width'       => 530,
    'height'      => 1000,
    'active_callback' => 'smoy_callback_is_front_page_or_eng_page'
)));
```

Esimerkkikoodi 22. Mukautinkentän lisäys teemamukauttimen taustakuvaosioon. Kentän kautta sivustolle ladataan rajattu kuvatiedosto.

Esimerkkikoodissa 22 huomionarvoinen asetusparametri on asetusoliolle määritettävä sanitize_callback-kenttä. Tämän kentän arvoksi asetettua funktiota kutsutaan asetuskentän datan suodattamiseen. Tällä varmistetaan, että syötedata on juuri odotettua tyyppiä, sillä täysin mielivaltaisen datan vastaanottaminen sovelluskoodin prosessoitavaksi on merkittävä tietoturvariski. Koska esimerkkikoodin 22 tapauksessa syötedatan oletetaan olevan kuvaolion tunnusluku, sopiva sanitointifunktio olisi absint, sillä se muuntaa suodatettavan datan positiiviseksi kokonaisluvuksi. Hyödyntämällä WP_Customize_Media_Control-luokkaa ja tästä luokasta periytyviä lapsiluokkia, teemamukauttimen kontrolleihin on mahdollista liittää WordPressin medianhallintarajapinnan käyttöliittymätoiminnallisuuksia. Tämä mahdollistaa sen, että käyttäjä voi ladata ja valita erilaisia mediatiedostoja sivuston mediakirjastosta teemamukauttimen asetuksen kautta. WP_Customize_Cropped_Image_Control-luokan liittäminen mukautinkontrolliin on erityisen hyödyllistä tilanteessa, jossa käyttäjän lataama kuva tulee rajata tiettyyn kuvasuhteeseen.

Projektissa sovellettiin WP_Customize_Cropped_Image_Control-luokkaa etusivun taustakuvien mukautinasetusten määrittämisessä, jotta sivustolle ladatuista leveistä taustakuvista olisi mahdollista rajata kapeilla mobiilinäytöillä esitettävä versio alkuperäisestä kuvasta. Esimerkkikoodissa 23 on esitetty lyhennelmä projektissa sovelletusta PHP-koodista, joka generoi dynaamiset CSS-tyylit etusivun osion taustakuvan esittämistä varten.

```
$css['#about-us']['background-image'] = "url(\"".$desktop_src."\")";

$css_media_query['#about-us']['background-image'] = "url(\"".$mo-
bile_src."\")";

if(!empty($css) && !empty($css_media_query)) {
  $final_css = '<style type="text/css">';
  foreach ( $css as $style => $style_array ) {
    $final_css .= $style . '{';
    foreach ( $style_array as $property => $value ) {
      $final_css .= $property . ':' . $value . ';';
    }
    $final_css .= '}';
  }
  $final_css .= '@media only screen and (max-width: 630px) {';
  foreach ( $css_media_query as $style => $style_array ) {
    $final_css .= $style . '{';
    foreach ( $style_array as $property => $value ) {
      $final_css .= $property . ':' . $value . ';';
    }
    $final_css .= '}';
  }
  $final_css .= '}';
  $final_css .= '</style>';
  echo $final_css;
}
```

Esimerkkikoodi 23. Taustakuvan tyylisäännöt generoidaan dynaamisesti.

Esimerkkikoodissa $desktop_src- ja $mobile_src-muuttujiin on tallennettu teemamukauttimen asetuskenttään tallennetun kuvaolion pöytäkone- ja mobiiliversioiden URL-osoitteet. Edeltävä koodi on sisällytetty wp_head-toimintakoukkuun kytkettyyn funktioon, joka tulostaa generoidut tyylit etusivulle head-osioon. Koodin tuottama mediakyselylohko esittää etusivun sisältöelementin taustakuvana alkuperäisestä kuvasta rajatun, kapeamman mobiiliversion, kun selainikkunan leveys on pienempi tai yhtä suuri kuin 630 pikseliä.

Teemamukautinrajapintaa sovellettiin projektissa laajamittaisesti erilaisten etusivun sisältöelementtien ja tyylien dynaamiseen luontiin. Toisena esimerkkinä mainittakoon etusivun otsakeosion taustavideo, jolle syötetään MP4- ja WEBM-formaatin videotiedostot teemamukauttimen asetuskontrollien kautta. Tämän videoelementin HTML5-merkintä ja mahdolliset CSS-tyylitykset ja JavaScript-komponentin käyttämät asetusparametrit luo-

daan dynaamisesti teemamukauttimen asetusdatan perusteella. Teemamukautinrajapinnan lisäksi muita merkittäviä projektissa sovellettuja komponentteja olivat mukautetut artikkelityypit ja vimpaimet. Yrityksen henkilöstöön liittyvää dataa varten kirjoitettiin oma mukautettu artikkelityyppi. Yrityksen työntekijän perustiedot ja viittaus henkilökuvaan tallennetaan tietokantaan, ja tiedot sekä henkilökuva esitetään etusivulla responsiivisessa ruudukossa.

Sivuston animoitu uutiskirjelomake ja kelluvat sosiaalisen median linkkipainikkeet toteutettiin käyttäen räätälöityjä vimpainluokkia. Sosiaalisen median painikkeiden toteutus ei edellyttänyt erillisen JavasScript-koodin kirjoittamista, kun taas sen sijaan uutiskirjelomakkeen ja asiakasreferenssiruudukon responsiivisia tyylejä, animaatioita ja logiikkaa varten kirjoitettiin suhteellisen runsaasti JavaScript-koodia. Lisäksi sivuston mobiiliystävällisen navigaatiopalkin takana on JavaScript-komponentti, joka säätelee navigaation tyylejä perustuen käyttäjän tekemiin toimintoihin sekä selainikkunan leveyden, korkeuden ja vierityspalkin paikan muutoksiin.

Lopullisten selaintestauksien, korjauksien ja tarkastuksen jälkeen annettiin hyväksyntä julkaista uusi teema yrityksen vuokraamalle verkkopalvelimelle. Uusi teema valmisteltiin tuotantoympäristöön julkaisua varten, ja vanhasta sivustosta sekä tietokannasta tallennettiin varmuuskopiot ennen siirtoa. Merkittävä osa valmisteluvaihetta oli useiden vanhan sivuston sivuosoitteiden uudelleenohjausten määrittäminen teeman functions.php-tiedostoon ja WordPress-asennuksen .htaccess-tiedostoon. Tämän avulla vältyttiin siltä, että Googlen kautta haettuja vanhoja sivuja linkitetään asianmukaisesti vastaaville uusille sivuille, eikä yrityksen hakukonenäkyvyys kärsisi 404-osumien myötä.

6 Yhteenveto

Opinnäytetyössä tarkasteltiin WordPress-julkaisujärjestelmää verkkosovelluskehyksenä ja arvioitiin alustan soveltuvuutta modernien verkkosivustojen kehittämiseen. Lisäksi työssä perehdyttiin useisiin verkkokehittämisen ja tarkemmin WordPress-kehittämisen menettelytapoihin, jotka voidaan perustella hyviksi käytänteiksi. Erityisesti työssä esitettyjen verkko-ohjelmointiin liittyvien käytänteiden soveltamisen keskeisenä motiivina on tuottaa helpommin ylläpidettävää, modulaarisempaa ja vähemmän riskialtista sovelluskoodia. Raportin tarkoitus ei ollut toimia perusteellisena listauksena WordPress-alustaan

sisäänrakennetuista rajapinnoista, metodeista ja muista sovelluskehityksessä hyödynnettävistä työkaluista. Sen sijaan olennainen tavoite oli havainnollistaa, että on perusteltua nykyään lähestyä WordPressiä ammattimaisen verkkosovelluskehityksen näkökulmasta, huolimatta alustan varsin syvälle juurtuneesta maineesta blogijulkaisujärjestelmänä.

Raportissa esitettyjä käytänteitä pyrittiin laajamittaisesti soveltamaan opinnäytetyön käytännön projektityössä, jossa asiakasyritykselle rakennettiin ja toimitettiin yrityksen tarpeiden pohjalta räätälöity WordPress-teema. Hyödyntämällä WordPressin tarjoamia rajapintoja teemakehityksessä projektissa saavutettiin asiakkaan näkökulmasta keskeinen tavoite: lähestulkoon mikään yrityksen sivuston asiakaspäässä esitettävästä informaatiosta tai verkkoaineistosta ei ole staattista, vaan datan editointiin on löydettävissä asianmukaiset asetusnäkymät ja -kentät. Projektin lopputuotteen potentiaalisia kehittämiskohteita olisi teeman PHP-koodin refaktorointi enemmän komponentti- ja oliopohjaiseen suuntaan. Tämä tarkoittaa käytännössä sitä, että teeman functions.php-tiedostoa kevennetään jakamalla tiedoston sisältö pienempiin kokonaisuuksiin ja tarvittaessa monimutkainen sovelluslogiikka koteloidaan erilaisten yleishyödyllisten ja uudelleenkäytettävien luokkien sisään.

Opinnäytetyö oli kokonaisuutena suhteellisen laajamittainen oppimisprosessi, jonka aikana kerrytettiin runsaasti WordPress-kehittämisen ja verkkosivustojen suunnittelun ja tuotannon tietotaitoja. WordPress on varteenotettava verkkosovelluskehys näyttävien pienten ja keskisuurten yritysten verkkosivustojen tuotantoon. WordPress-dokumentaation läpikotainen opiskelu ei yksistään kuitenkaan tee ammattimaista WordPress-kehittäjää; tämä edellyttää laajemmassa mittakaavassa verkon eri teknologioiden, protokollien ja palvelininfrastruktuurien syvää tuntemusta.

Lähteet

1 Web Content Management System. 2008. Verkkoaineisto. The Government of the Hong Kong Special Administrative Region. <https://www.infosec.gov.hk/english/technical/files/web.pdf>. Luettu 5.3.2018.

2 Introduction to Content Management Systems. 2010. Verkkoaineisto. Hannon Hill Corporation. <http://web.csulb.edu/committees/webcomm/hannonhill/Hannon_Hill_Content_Management_White_Paper.pdf>. Luettu 5.3.2018.

3 Marohnić, Viktor. 2014. What we can learn from the evolution of Content Management Systems. Verkkoaineisto. The Next Web. <https://thenextweb.com/dd/2014/02/20/can-learn-evolution-content-management-systems/#.tnw_Cp5V56Li#.tnw_LTmX8kfh>. Julkaistu 20.2.2014. Luettu 5.3.2018.

4 Yank, Kevin. 2001. Which Server-Side Language Is Right For You? Verkkoaineisto. Sitepoint. <https://www.sitepoint.com/server-side-language-right/>. Julkaistu 9.10.2001. Luettu 5.3.2018.

5 Jenkins, Tom. 2012. Behind the Firewall. Big Data and the Hidden Web: The Path to Enterprise Information Management. Verkkoaineisto. Open Text Corporation. <http://2015.leadershipcanada.ca/wp-content/uploads/2015/04/Behind-The-Firewall-Big-Data-and-the-Hidden-Web-Tom-Jenkins-April-2012.pdf>. Luettu 5.3.2018.

6 Server-side web frameworks. 2017. Verkkoaineisto. MDN web docs. <https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Web_frameworks>. Muokattu 1.9.2017. Luettu 12.10.2017.

7 Fitch-Cook, Elisha. 2015. A Brief History of Web Publishing. Verkkoaineisto. <https://www.cooper.com/journal/2015/3/a-brief-history-of-web-publishing>. Julkaistu 24.3.2015. Luettu 5.3.2018.

8 Brampton, Martin. 2010. PHP 5 CMS Framework Development. E-kirja. Packt Publishing.

9 Olusola, Maitanmi & Sunday, Idowu. 2013. Evaluation of Content Management Systems Performance. Verkkoaineisto. <http://www.iosrjournals.org/iosr-jce/papers/Vol9-Issue4/K0946269.pdf?id=251>. Luettu 13.10.2017.

10 Bramscher, Paul & Butler, John. 2005. LibData to LibCMS, One library's evolutionary pathway to a content management system. Library Hi Tech, Content management systems. E-kirja. Emerald Group Publishing.

11 Usage of content management systems for websites. 2018. Verkkoaineisto. W3Techs.com, Q-Success. <https://w3techs.com/technologies/overview/content_management/all>. Luettu 7.3.2018.

12 Open source software is taking over. 2016. Verkkoaineisto. Wiredelta. <https://blog.wiredelta.com/open-source-software-taking-over/>. Julkaistu 21.9.2016. Luettu 3.3.2018.

13 Cid, Daniel. 2016. Hacked Website Report – 2016/Q3. Verkkoaineisto. Sucuri Inc. <https://blog.sucuri.net/2017/01/hacked-website-report-2016q3.html>. Julkaistu 4.1.2016. Luettu 3.3.2018.

14 Jones, Kyle & Farrington Polly-Alida. 2013. Learning from Libraries that Use WordPress, Content-Management System Best Practices and Case Studies. E-kirja. American Library Association.

15 Ratnayake, Rakhitha. 2013. WordPress Web Application Development. E-kirja. Packt Publishing.

16 Simple Example of MVC (Model View Controller) Design Pattern for Abstraction. 2008. Verkkoaineisto. CodeProject. <https://www.codeproject.com/Articles/25057/Simple-Example-of-MVC-Model-View-Controller-Design>. Luettu 18.10.2017.

17 McFarlin, Tom. 2013. Design Patterns in WordPress: An Introduction. Verkkoaineisto. Envato Tuts+. <https://code.tutsplus.com/articles/design-patterns-in-wordpress-an-introduction--wp-31604>. Julkaistu 10.5.2013. Luettu 24.10.2017.

18 Pollock, Josh. 2016. Basic PHP Design Patterns for WordPress Developers. Verkkoaineisto. Torque. <https://torquemag.io/2016/06/basic-php-design-patterns-developers>. Julkaistu 21.6.2016. Luettu 24.10.2017.

19 Theme Development. Verkkoaineisto. Codex, WordPress.org. <https://codex.wordpress.org/Theme_Development>. Luettu 26.10.2017.

20 Onishi, Adam. 2013. Pro WordPress Theme Development. E-kirja. Apress.

21 Query Overview. Verkkoaineisto. Codex, WordPress.org. <https://codex.wordpress.org/Query_Overview>. Luettu 30.10.2017.

22 Class Reference/WP Query. Verkkoaineisto. Codex, WordPress.org. <https://codex.wordpress.org/Class_Reference/WP_Query>. Luettu 30.10.2017.

23 Template Hierarchy, Theme Handbook. Verkkoaineisto. Developer Resources, WordPress.org. <https://developer.wordpress.org/themes/basics/template-hierarchy/>. Luettu 30.10.2017.

24 Hayes, David. 2015. Child Themes, the Template Hierarchy, and One Great Little Hack. Verkkoaineisto. WPShout. <https://wpshout.com/child-themes-the-template-hierarchy-and-one-sanity-saving-hack/>. Julkaistu 7.4.2015. Luettu 31.10.2017.

25 Code Reference, add_action(). Verkkoaineisto. Developer Resources, WordPress.org. <https://developer.wordpress.org/reference/functions/add_action/>. Luettu 2.11.2017.

26 Code Reference, noindex(). Verkkoaineisto. Developer Resources, WordPress.org. <https://developer.wordpress.org/reference/functions/noindex/>. Luettu 2.11.2017.

27 Code Reference, wp_no_robots(). Verkkoaineisto. Developer Resources, WordPress.org. <https://developer.wordpress.org/reference/functions/wp_no_robots/>. Luettu 2.11.2017.

28 Maier, Thomas. 2016. Useful Tips To Get Started With WordPress Hooks. Verkkodokumentti. Smashing Magazine. <https://www.smashingmagazine.com/2016/01/get-started-with-hooks-wordpress/>. Julkaistu 5.1.2016. Luettu 2.11.2017.

29 Plugin API/Filter Reference/the content. Verkkoaineisto. Codex, WordPress.org. <https://codex.wordpress.org/Plugin_API/Filter_Reference/the_content>. Luettu 6.11.2017.

30 Code Reference, apply_filters(). Verkkoaineisto. Developer Resources, WordPress.org. <https://developer.wordpress.org/reference/functions/apply_filters/>. Luettu 6.11.2017.

31 Function Reference/remove filter. Verkkoaineisto. Codex, WordPress.org. <https://codex.wordpress.org/Function_Reference/remove_filter>. Luettu 9.11.2017.

32 Function Reference/remove action. Verkkoaineisto. Codex, WordPress.org. <https://codex.wordpress.org/Function_Reference/remove_action>. Luettu 9.11.2017.

33 McCollin, Rachel. 2014. Understanding and Working With Data in WordPress. Verkkoaineisto. Envato Tuts+. <https://code.tutsplus.com/tutorials/understanding-and-working-with-data-in-wordpress--cms-20567>. Julkaistu 28.7.2014. Luettu 6.3.2018.

34 Zahari, Firdaus. 2016. Working with Databases in WordPress. Verkkoaineisto. Sitepoint. <https://www.sitepoint.com/working-with-databases-in-wordpress/>. Julkaistu 20.1.2016. Luettu 6.3.2018.

35 Class Reference/wpdb. Verkkoaineisto. Codex, WordPress.org. <https://codex.wordpress.org/Class_Reference/wpdb>. Luettu 6.3.2018.

36 Vagrant Documentation. Verkkoaineisto. HashiCorp. <https://www.vagrantup.com/docs/>. Luettu 15.11.2017.

37 Stubin, Toni. 2015. Amazon Web Services perustaa toimiston Suomeen. Verkkoaineisto. Viestintätoimisto Netprofile. <http://netprofile.fi/tiedotteet/amazon-web-services-perustaa-toimiston-suomeen/>. Julkaistu 22.10.2015. Luettu 21.11.2017.

38 Amazon EC2 Instance Types. Verkkoaineisto. Amazon Web Services. <https://aws.amazon.com/ec2/instance-types/>. Luettu 27.11.2017.

39 Chatzakis, Andreas. 2014. WordPress: Best Practices on AWS, Reference Architecture for Scalable WordPress-powered Websites. Verkkodokumentti. Amazon Web Services. <https://d0.awsstatic.com/whitepapers/wordpress-best-practices-on-aws.pdf>. Luettu 27.11.2017.

40 Version Control Systems Popularity in 2016. Verkkoaineisto. RhodeCode. <https://rhodecode.com/insights/version-control-systems-2016>. Luettu 9.12.2017.

41 Chacon, Scott & Straub, Ben. 2014. Pro Git, Everything you need to know about Git. Second Edition. E-kirja. Apress.

42 Kearney, Meggin. 2017. Measure Performance with the RAIL Model. Verkko-aineisto. Google Developers. <https://developers.google.com/web/fundamentals/performance/rail>. Muokattu 26.9.2017. Luettu 13.12.2017.

43 Perna, Maria Antonietta. 2017. Is Using SVG Images Good for Your Website's Performance? Verkkoaineisto. SitePoint. <https://www.sitepoint.com/svg-good-for-website-performance/>. Julkaistu 27.3.2017. Luettu 16.12.2017.

44 Giltsoff, Jake. SVG On the Web – A Practical Guide. Verkkoaineisto. <https://svgontheweb.com/>. Luettu 16.12.2017.

45 Ellingwood, Justin. 2015. Web Caching Basics: Terminology, HTTP Headers, and Caching Strategies. Verkkoaineisto. DigitalOcean. <https://www.digitalocean.com/community/tutorials/web-caching-basics-terminology-http-headers-and-caching-strategies>. Julkaistu 1.4.2015. Luettu 20.12.2017.

46 Kundu, Sourav. 2014. How does WordPress Caching Work? Verkkoaineisto. WPExplorer.com. <http://www.wpexplorer.com/wordpress-caching-work/>. Muokattu 3.3.2014. Luettu 23.12.2017.

47 Class Reference/WP Object Cache. Verkkoaineisto. Codex, WordPress.org. <https://codex.wordpress.org/Class_Reference/WP_Object_Cache>. Luettu 23.12.2017.

48 Speed Up Dynamic Pageviews, Persistent Object Caching, aka The WordPress Turbo Button. 2016. Verkkoaineisto. WordPress at Scale. <https://github.com/pantheon-systems/wordpress-at-scale/blob/master/_pages/object-caching.md>. Muokattu 8.2.2016. Luettu 23.12.2017.

49 Transients API. Verkkoaineisto. Codex, WordPress.org. <https://codex.wordpress.org/Transients_API>. Luettu 23.12.2017.

50 Gooding, Sarah & Hellyer, Ryan. 2014. Persistent Object Caching. Verkkoaineisto. WordPress Tavern. <https://wptavern.com/persistent-object-caching>. Julkaistu 7.11.2014. Luettu 23.12.2017.

51 Morales, David. 2016. What is Memcached? Verkkoaineisto. <https://davidmles.com/what-is-memcached/>. Julkaistu 13.1.2016. Luettu 23.12.2017.

52 Graham, Geoff. 2015. The Difference Between Responsive and Adaptive Design. Verkkoaineisto. CSS-Tricks. <https://css-tricks.com/the-difference-between-responsive-and-adaptive-design/>. Julkaistu 11.11.2015. Luettu 24.12.2017.

53 Van Hove, Niels. What are CSS Media Queries and how to implement them. Verkkoaineisto. CSS Media Queries. <http://cssmediaqueries.com/what-are-css-media-queries.html>. Luettu 30.12.2017.

54 CSS, Overview. Verkkoaineisto. Bootstrap. <https://getbootstrap.com/docs/3.3/css/>. Luettu 30.12.2017.

55 Liew, Zell. 2014. How To Write Mobile-first CSS. Verkkoaineisto. Zell. <https://zellwk.com/blog/how-to-write-mobile-first-css/>. Julkaistu 17.12.2014. Luettu 30.12.2017.

56 Kay, Mickey. 2013. What is Mobile First CSS and Why Does It Rock? Verkkoaineisto. MIGHTYminnow. <https://www.mightyminnow.com/2013/11/what-is-mobile-first-css-and-why-does-it-rock/>. Julkaistu 8.11.2013. Luettu 30.12.2017.

57 Twentysixteen/functions.php. 2016. Verkkoaineisto. Github. <https://github.com/WordPress/twentysixteen/blob/master/functions.php>. Muokattu 27.7.2016. Luettu 4.1.2018.

58 Code Reference, wp_get_attachment_image_attributes. Verkkoaineisto. Developer Resources, WordPress.org. <https://developer.wordpress.org/reference/hooks/wp_get_attachment_image_attributes/>. Luettu 4.1.2018.

59 McGill, Joe. 2015. Responsive Images: Merge Proposal. Verkkoaineisto. WordPress.org. <https://make.wordpress.org/core/2015/09/30/responsive-images-merge-proposal/>. Julkaistu 30.9.2015. Luettu 4.1.2018.

60 Evko, Tim. 2015. Responsive Images Now Landed In WordPress Core. Verkkoaineisto. Smashing Magazine. <https://www.smashingmagazine.com/2015/12/responsive-images-in-wordpress-core/>. Julkaistu 24.12.2015. Luettu 4.1.2018.

61 Nelson, Joshua David. 2016. Using WordPress responsive images for css background-image property, in-line styling. Verkkoaineisto. <https://gist.github.com/joshuadavidnelson/eb4650aa1ee8da9c7d731960e9402e21>. Muokattu 12.4.2016. Luettu 6.1.2018.

62 Code Reference, get_single_template(). Verkkoaineisto. Developer Resources, WordPress.org. <https://developer.wordpress.org/reference/functions/get_single_template/>. Luettu 7.1.2018.

63 Template Tags. Theme Handbook. Verkkoaineisto. Developer Resources, WordPress.org. <https://developer.wordpress.org/themes/basics/template-tags/>. Luettu 15.1.2018.

64 Function Reference/the date. Verkkoaineisto. Codex, WordPress.org. <https://codex.wordpress.org/Function_Reference/the_date>. Luettu 22.1.2018.

65 Including CSS & JavaScript. Theme Handbook. Verkkoaineisto. Developer Resources, WordPress.org. <https://developer.wordpress.org/themes/basics/including-css-javascript/>. Luettu 22.1.2018.

66 Function Reference/wp localize script. Verkkoaineisto. Codex, WordPress.org. <https://codex.wordpress.org/Function_Reference/wp_localize_script>. Luettu 25.1.2018.

67 Code Reference, wp_register_script. Verkkoaineisto. Developer Resources, WordPress.org. <https://developer.wordpress.org/reference/functions/wp_register_script/>. Luettu 23.1.2018.

68 Widgets. Theme Handbook. Verkkoaineisto. Developer Resources, WordPress.org. <https://developer.wordpress.org/themes/functionality/widgets/>. Luettu 27.1.2018.

69 Function Reference/register sidebar. Verkkoaineisto. Codex, WordPress.org. <https://codex.wordpress.org/Function_Reference/register_sidebar>. Luettu 27.1.2018.

70 PHP Coding Standards. 2014. Verkkoaineisto. Make WordPress Core, WordPress.org. <https://make.wordpress.org/core/handbook/best-practices/coding-standards/php/>. Muokattu 20.10.2014. Luettu 3.2.2018.

71 Hayes, David. 2017. Understanding PHP Output Buffering, and Why It's Great for Shortcodes. Verkkoaineisto. WPShout.<https://wpshout.com/php-output-buffering/>. Julkaistu 15.8.2017. Luettu 3.2.2018.

72 Output Buffering Control. Verkkoaineisto. PHP.net. <http://php.net/manual/en/book.outcontrol.php>. Luettu 3.2.2018.

73 Closures. 2018. Verkkoaineisto. MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures>. Muokattu 2.2.2018. Luettu 10.2.2018.

74 JavaScript Coding Standards. Verkkoaineisto. Make WordPress Core, WordPress.org. <https://make.wordpress.org/core/handbook/best-practices/coding-standards/javascript/>. Luettu 12.2.2018.